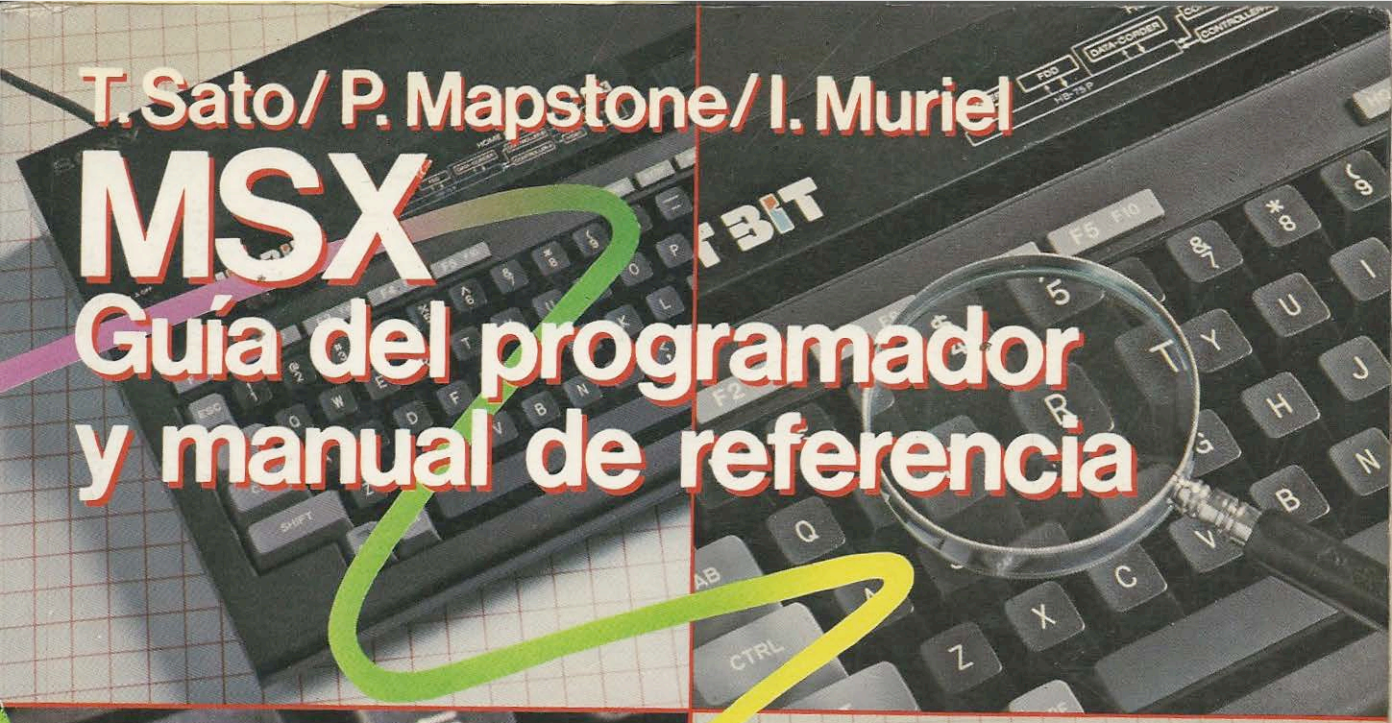
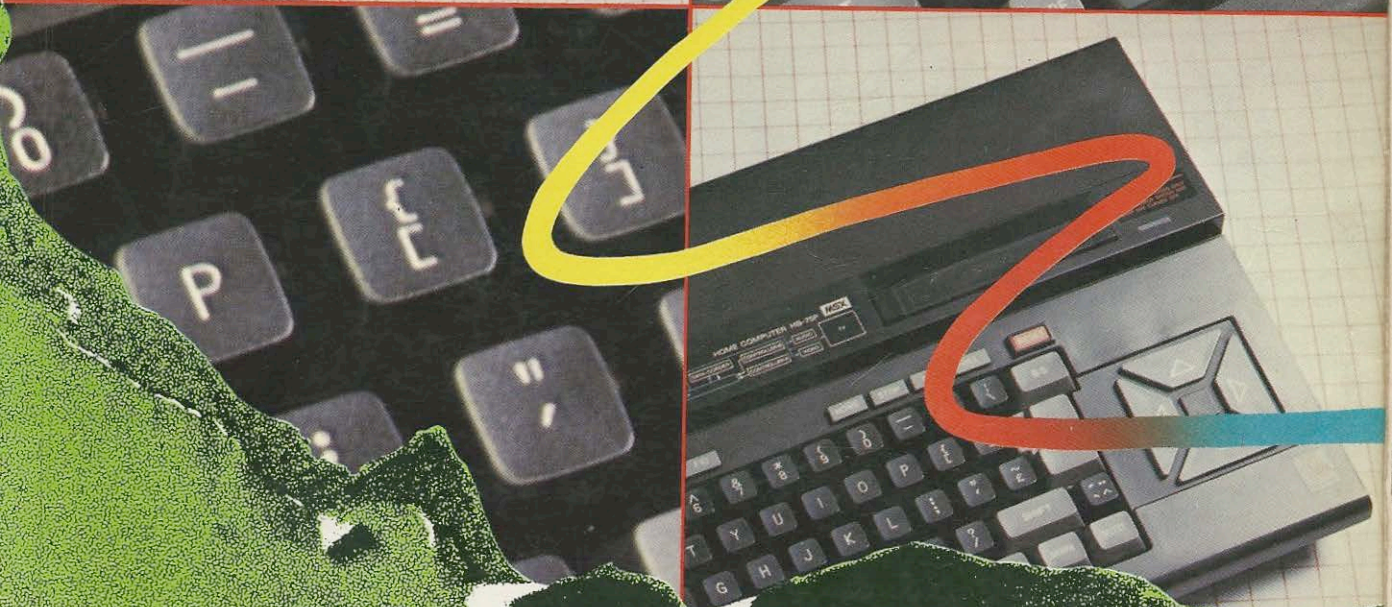
T. Sato / P. Mapstone / I. Muriel

MSX

Guía del programador y manual de referencia





VIFI
ANAYA

MSX-Guía del programador y manual de referencia

Toshiyuki Sato
Paul Mapstone
Isabella Muriel

ANAYA MULTIMEDIA

MICROINFORMATICA

Título de la obra original:
THE COMPLETE MSX PROGRAMMERS GUIDE

Traducción: Celso Losada, Juan Carlos Contreras y Juan José Rodríguez
Diseño de colección: Antonio Lax
Diseño de cubierta: Narcís Fernández

Reservados todos los derechos. Ni la totalidad ni parte de este libro puede reproducirse o transmitirse por ningún procedimiento electrónico o mecánico, incluyendo fotocopia, grabación magnética o cualquier almacenamiento de información y sistema de recuperación, sin permiso escrito de Ediciones Anaya Multimedia, S. A.

© 1984 Toshiyuki Sato, Paul Mapstone, Isabella Muriel

Edición publicada por acuerdo con
Melbourne House (Publishers) Ltd.,
Londres.

© EDICIONES ANAYA MULTIMEDIA, S. A., 1985
Villafranca, 22. 28028 Madrid
Depósito legal: M. 39.943-1985
ISBN: 84-7614-059-2
Printed in Spain
Imprime: Librograf
Molina Seca, 13. Fuenlabrada (Madrid)

Indice

PARTE PRIMERA

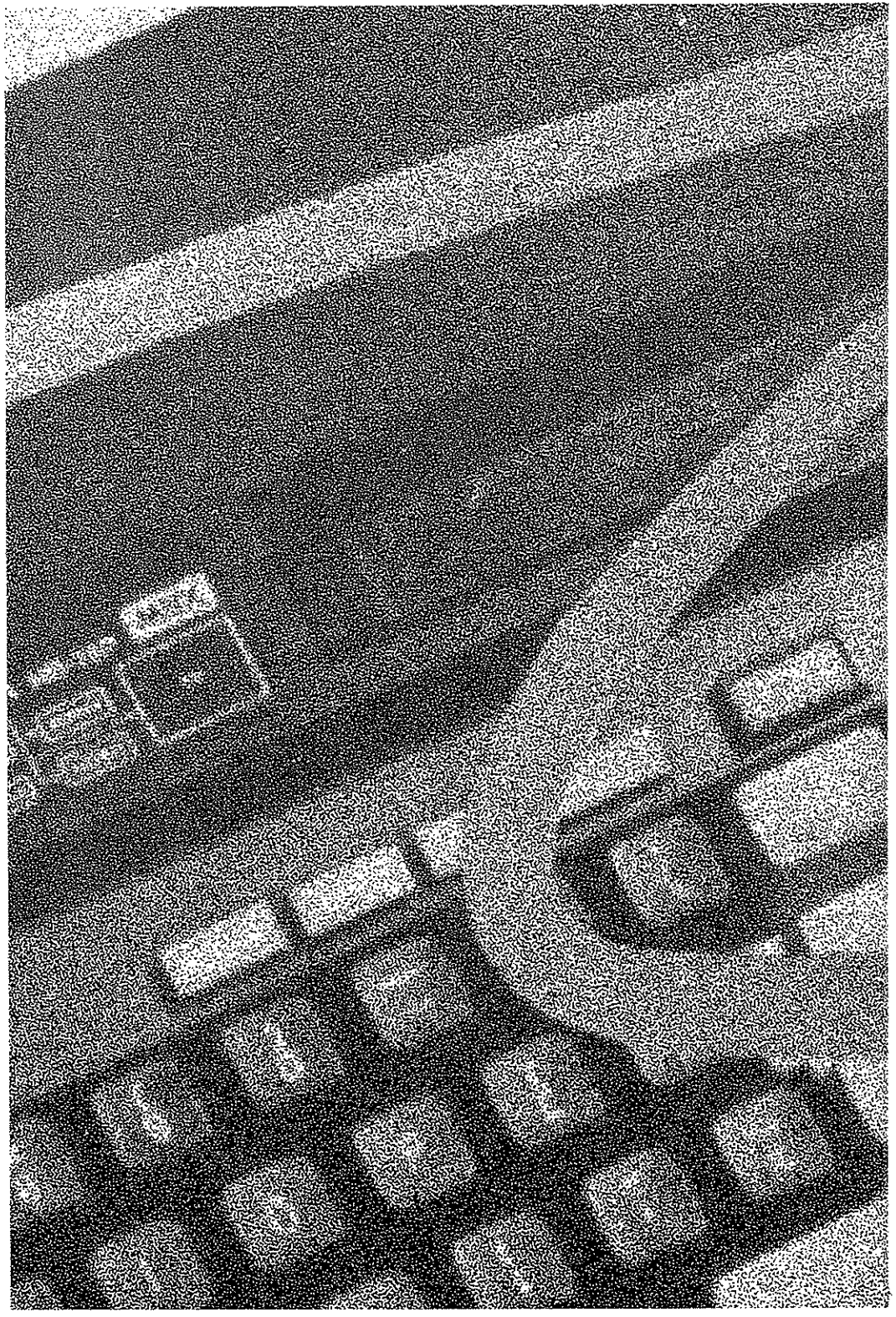
INTRODUCCION AL BASIC DEL MSX

PARTE SEGUNDA

GUIA AVANZADA DE PROGRAMACION

PARTE PRIMERA

INTRODUCCION AL BASIC DEL MSX



1

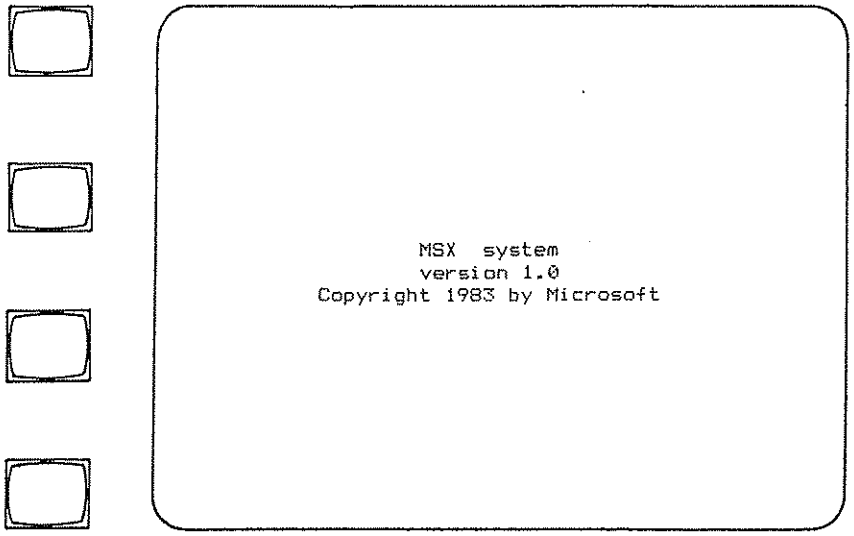
Introducción

Al desempaquetar tu ordenador MSX encontrarás junto a él un cable de conexión de video. Este cable tiene una conexión PHONO en ambos extremos que se emplea para conectar el ordenador a la televisión.

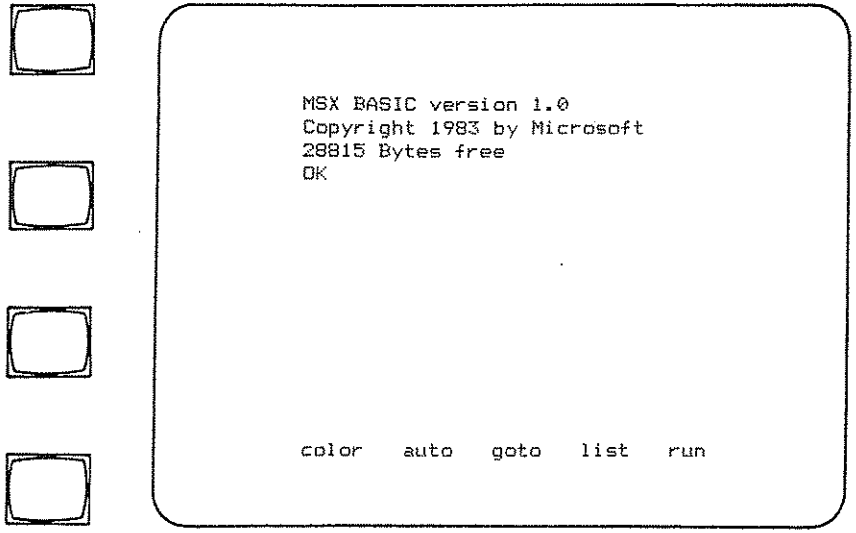
Por el momento, sólo necesitas conectar el ordenador a la televisión; cualquiera te sirve. Si posees un televisor de color, tienes la ventaja de poder utilizar el comando de color del MSX, pero también se puede emplear uno de blanco y negro.

Para conectarlo a la TV, introduce una de las conexiones PHONO en la toma que pone "TV" situada en la parte trasera del ordenador, y la otra en la antena correspondiente del televisor. Si tu aparato tiene dos entradas en las que pone UHF y VHF, utiliza la de UHF.

Ahora enciende el televisor y el ordenador. Primero el televisor, y espera hasta que se caliente; luego enciende el ordenador. Si la sintonización es correcta, lo primero que verás en pantalla es



y, después de una pausa, ésta otra:



Si no ves estas pantallas, entonces sintoniza tu televisor hasta que las veas. (El número de bytes libres depende de la cantidad de memoria que tenga tu máquina.) Esto ocurre cada vez que se pone en marcha el MSX.

Si la pantalla de tu televisor varía continuamente la imagen, ajústala hasta que obtengas en ella una claridad total. Si tienes un botón para cada canal es preferible que sintonices uno que esté libre.

Ahora ya estás preparado para trabajar con el teclado. Experimenta cuanto quieras, que no estropearás el ordenador. Todo lo que escribas aparecerá en la pantalla del televisor. Probablemente obtendrás mensajes de error, pero no te preocupes: aprenderás a no tenerlos.

Si miras a la pantalla, una vez en funcionamiento, verás un pequeño cuadrado blanco debajo del mensaje "OK". Es el cursor de texto, y te indica dónde se escribirá el siguiente carácter en pantalla. Prueba a escribir algunas letras.

Es mejor que te acostumbres a utilizar ambas manos cuando teclees. Pulsando cualquiera de las teclas de carácter aparecerá en pantalla ese carácter. Teclea:

Abcd Efgh Ijklm Nopq Rstu Vwxyz

Para espaciar pulsa la barra espaciadora, situada en la parte inferior del teclado. Para escribir en mayúsculas pulsa la tecla <SHIFT> y la tecla del carácter simultáneamente.

En la parte inferior izquierda del teclado se encuentra la tecla <CAPS LOCK>. Si pulsas esta tecla, todo lo que teclees a continuación se convertirá automáticamente en letras mayúsculas, y se encenderá, cerca de ella, una luz verde. Para salir, pulsa de nuevo la misma tecla.

Ahora mira los números. Observa lo que ocurre cuando pulsas <SHIFT> y un número del 1 al 9.

En la zona superior del teclado hay cinco teclas, que tienen escritas en ellas desde F1 hasta F10. Son las teclas de función. En la zona inferior de la pantalla se visualizan las funciones de estas teclas. Olvídalas por el momento, ya veremos sus funciones en el tema 8.

Busca la tecla <STOP>. La utilizarás mucho con la tecla de control <CTRL>. Por el momento la tecla <ESC> no hace nada, pero se empleará para controlar la impresora.

La tecla <TAB> mueve el cursor a través de la pantalla. Borra todo lo que encuentra en su camino. Si el cursor estaba originalmente a la izquierda, se mueve nueve posiciones. Si pulsas <TAB> nuevamente, se mueve hasta la posición 17, luego a la 25...

La tecla <GRAPH> te permite escribir caracteres gráficos, a los que accedes por medio de las teclas de caracteres. Pulsando <GRAPH> y la tecla de carácter correspondiente obtendrás un carácter gráfico.

Si pulsas <SHIFT> <GRAPH> y una tecla de carácter obtendrás otro carácter distinto. Busca los caracteres gráficos ♥, ♦, ♣ y ♠.

Pulsando <CODE> y una tecla de carácter obtendrás los caracteres europeos y griegos (por ejemplo, α, β, $\ddot{a}$, $\ddot{u}$, etc.). Experimenta con esta tecla. Inténtalo pulsando también <SHIFT>. La mayoría de las teclas te darán un carácter CODE diferente.

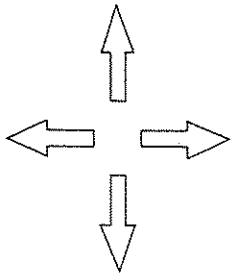
Una tecla muy importante es <RETURN>. Cuando quieras que el ordenador lea algo que has escrito en pantalla pulsa <RETURN>. Si la presionas ahora es probable que obtengas un mensaje de error. No te preocupes: significa que el ordenador no ha entendido lo que le has escrito.

En este momento la pantalla debe estar bastante llena; observa que cuando

llegas al final de la pantalla todo se mueve automáticamente hacia arriba una línea, y pierdes la superior. A esto se le denomina "desplazamiento" *(scrolling)*. Para limpiar la pantalla pulsa <SHIFT> y <HOME> simultáneamente. <HOME> es una de las cuatro teclas de edición; se encuentra en la parte superior derecha del teclado. Observa que después de pulsarla el cursor salta automáticamente a la parte superior izquierda de la pantalla. Presionando esta tecla sin <SHIFT> provocas que el cursor se posicione en la parte superior izquierda de la pantalla sin borrar el texto.

Ahora escribe una línea entera de caracteres. Para borrarla pulsa la tecla de retroceso de espacio, <BS>. Te borrará cualquier carácter a la izquierda del cursor. Como cualquiera de las otras teclas, se repite si la mantienes presionada; por tanto, lo más probable es que en este momento hayas borrado la línea entera.

Escribe más caracteres y experimenta con las siguientes teclas del cursor:



mueven el cursor en la dirección de la flecha sin borrar nada de lo que se encuentra debajo.

Intenta borrar un carácter en concreto utilizando <BS> y las teclas del cursor. Cuando creas que has captado la idea observa lo que sucede cuando pulsas <DEL> (otra de las cuatro teclas de edición).

Esta tecla borra el carácter en donde se encuentra situado el cursor y desplaza una posición todo lo que hay a su derecha. Si mantienes presionada la tecla <DEL> borrarás todo lo que había originalmente a la derecha del cursor.

Supongamos que tienes en pantalla lo siguiente:

```
abcfg
```

Si deseas insertar, por ejemplo, una "d" entre la "c" y la "f", mueve el cursor hasta situarlo sobre el carácter "f"; entonces pulsa la tecla de inserción <INS>. El cursor se sustituirá por una raya en blanco. Si ahora tecleas "d", aparecerá en la parte izquierda de la línea entre "c" y "f". Para poner en su estado inicial al cursor pulsa de nuevo <INS>. Practica utilizando el cursor y las teclas de edición.

La cuarta tecla de edición es <SELECT> y, por el momento, no hace nada.

Ejercicios

1. Escribe los dos caracteres gráficos asociados con cada tecla, así como el carácter de ésta. Ahora borra los caracteres alfanuméricos, dejando en pantalla sólo los caracteres gráficos.
2. Imprime el número 0 y la letra O. ¿Observas diferencia entre ambos? Cuando estés escribiendo un número debes teclear 0, ya que en caso contrario el ordenador te dará un error. Compara también el 1 con la letra ele mayúscula. No los confundas.

2

El modo directo

Cuando te hayas familiarizado con el uso de <BS> y las teclas de edición, estarás preparado para utilizar los comandos del ordenador. Primero limpia la pantalla, pulsando las teclas <SHIFT> y <CLS> simultáneamente. Realiza esta operación siempre que la pantalla se encuentre llena.

Ahora escribe con cuidado:

PRINT "BIENVENIDO AL MSX"

A continuación pulsa la tecla <RETURN>.

Verás "BIENVENIDO AL MSX" bajo la línea que acabas de introducir, con el mensaje "OK". Quizá hayas obtenido un mensaje de error. No te preocupes: repítelo de nuevo; probablemente te equivocaste al escribirlo.

Cuando pulsas <RETURN> el ordenador ejecuta un comando. Sólo reconoce algunas palabras, llamadas "palabras reservadas", una de las cuales es PRINT. Cuando introduces PRINT, seguido de algo entrecomillado, sabe que tiene que escribir todo aquello que esté entre comillas.

Al ordenador no le preocupa que trabajes con letras mayúsculas o minúsculas; también ignora los espacios. (Sin embargo, no introduzcas espacios entre una palabra clave, ya que el ordenador no la tomará como tal.)

print "Bienvenido al MSX" <RETURN>

El resultado será el mismo que antes.

Cualquier carácter que esté entre comillas será escrito de la misma manera en que tú lo hayas tecleado. A la línea:

```
"Bienvenido al MSX"
```

se le llama "cadena de caracteres". Cuando el ordenador reconoce una sentencia PRINT, seguida de una cadena de caracteres, elimina las comillas y escribe exactamente el resto de la cadena tal como esté. Más adelante hablaremos de las cadenas de caracteres.

Para escribir un número no es necesario utilizar las comillas. Escribe:

```
PRINT 12345 <RETURN>
```

Verás 12345 justo debajo de la línea en que previamente habías escrito la orden. Aquí tampoco importan los espacios que dejes entre la sentencia PRINT y el número. Los espacios que hayas puesto dentro del número a escribir también se omitirán.

```
PRINT 12 345 <RETURN>
```

da el mismo resultado que antes.

Ahora teclea:

```
PRINT 3+4  <RETURN>
```

El ordenador escribirá el valor 7. Ha evaluado la expresión 3 + 4 y proporciona el resultado. Si deseas que te escriba 3 + 4 tienes que convertirlo en cadena de caracteres, usando las comillas:

```
PRINT "3 + 4"  <RETURN>
```

El ordenador trata a 3 + 4 como una cadena de caracteres, y lo escribe exactamente tal como se ha escrito.

Para restar dos números utiliza el signo "−".

Para multiplicar dos números se utiliza el signo "*" en lugar del signo "×". Esto se hace para evitar confusiones con la letra "x".

Para dividir utiliza el signo "/" en vez de "÷".

Escribe en orden lo siguiente. No te olvides de pulsar <RETURN> después de cada línea; de ahora en adelante lo daremos por hecho.

```
PRINT 3 * 4
PRINT "3 + 4 =";3 + 4
```

En el último ejemplo obtendrás:

```
3 + 4 = 7
```

Cuando el ordenador reconoce un punto y coma sabe que tiene que escribir otro término, en este caso la expresión 3 + 4. Entonces evalúa la expresión y escribe el resultado a continuación de la cadena de caracteres dada.

Notarás que siempre se deja un espacio en blanco antes y después de un número positivo.

```
PRINT "aa" ; 3;4;−5 "bb""cc"
```

te dará

```
aa 3 4 −5 bbcc
```

Como puedes observar, no se dejan espacios cuando se escriben cadenas de caracteres. Cuando se escribe un número negativo, el signo se coloca en el espacio en blanco que le precede.

Si no se pone ninguna puntuación entre las cadenas de caracteres, el ordenador asume automáticamente un punto y coma. Debes poner signos de puntuación entre los números; en caso contrario el ordenador lo tomará como un único número.

El ordenador también reconoce la coma en la sentencia PRINT; en dicho caso escribe los elementos en la misma línea, pero el segundo elemento lo escribirá quince posiciones más adelante del que se había escrito en primer lugar.

Escribe las siguientes líneas:

```
LET A = 3
PRINT A
```

Verás en pantalla el número 3.

El comando LET asigna un valor a una variable (en nuestro caso es A). Cuando el ordenador recibe el comando PRINT espera un número, ya que no hay comillas. Entonces busca en memoria y encuentra el número que se le ha asignado a la variable A, y es el que escribe.

Puedes tener expresiones más complicadas en el comando LET:

```
LET B = 31 * 72 * 4
PRINT B
```

Se visualiza el número 8928. Ahora escribe:

```
LET B = B + 1000
```

Obtendrás:

```
9928
```

La sentencia LET arriba indicada es incorrecta matemáticamente, pero es

totalmente correcta para el ordenador. Se evalúa la expresión a la derecha del signo "=" y su resultado se asigna a la variable que hay a la izquierda. En este caso, la variable B es igual a su antiguo valor, + 1000.

Los nombres de las variables deben empezar con una letra. Después se pueden poner números. Los espacios en blanco dentro del nombre de una variable se ignorarán.

```
A A es equivalente a AA
```

Puedes emplear letras mayúsculas y minúsculas, pero el ordenador no hace distinción entre ellas.

```
ABCO es igual que abco
```

Es importante no incluir una palabra reservada en el nombre de una variable:

```
MONO
```

no es válida, ya que ON es una palabra reservada.

Algunos términos sólo se pueden utilizar como palabras reservadas y nunca como nombre de variable; por ejemplo:

```
NAME
```

no es válida, ya que "NAME" es una palabra reservada.

Aquí tienes unos nombres de variables permitidos:

```
número2
n2
NUM
```

Los siguientes nombres de variables son incorrectos:

2n	El primer carácter es un dígito.
num?	"?" no es un carácter válido en el nombre de una variable.

También se puede asignar a las variables cadenas de caracteres:

```
LET A$ = "Hola"
PRINT A$
```

Verás "Hola". En este ejemplo, PRINT A$ es lo mismo que poner PRINT "Hola". Los nombres de las variables de tipo cadena de caracteres terminan con el signo "$" para distinguirlos de las variables numéricas. Por lo demás, tienen las mismas especificaciones que las variables numéricas.

Aquí tienes algunos nombres válidos de variables de tipo cadena de caracteres:

```
DIA$
CALLE$
L1$
```

Estos no son válidos:

A	No tiene el signo $.
A.B$	"." no es un carácter válido.
1PRIME$	El primer carácter debe ser una letra.
ON$	ON es una palabra reservada.
NAME$	NAME es una palabra reservada.
TONO$	ON es una palabra reservada.

Otro comando muy útil es INPUT. Escribe:

```
INPUT A
```

Cuando el ordenador ejecuta la instrucción visualiza una interrogación y espera que introduzcas un número. Este número se asigna a la variable A. Teclea un número cualquiera, verás cómo aparece junto con la interrogación. Si ahora escribes:

```
PRINT A
```

el número que acabas de introducir se visualizará. Si tecleas una cadena de caracteres se producirá un error.

Ejercicio

1. Qué valor tiene A en la siguiente expresión:
   A = 49 * 38.07/13

3

Escritura de un programa

Ahora ya has trabajado con el modo comando; has escrito un comando en pantalla, pulsado <RETURN> y el ordenador lo ha ejecutado. Si deseas usar de nuevo el mismo comando, deberás volver a escribirlo, lo que consume mucho tiempo.

Pero si deseas que el ordenador memorice el comando, lo que debes hacer es darle un número de línea; por ejemplo:

```
10  PRINT "ESTO ES BONITO"  <RETURN>
```

Cuando pulsaș <RETURN> el ordenador advierte la existencia del número de línea, relacionando ésta con un programa. Por tanto, almacena la línea en memoria y visualiza "OK" en pantalla.

Para que el ordenador ejecute la línea en cuestión debes utilizar el comando RUN:

```
RUN  <RETURN>
```

visualizándose

```
ESTO ES BONITO
```

Para escribir un programa, cada comando debe ir precedido por un número. Este número puede alcanzar un valor hasta 65529; lo mejor es empezar por el 10 y seguir de 10 en 10, es decir, tener la siguiente secuencia: 10, 20, 30, etc.

De esta manera, cuando hayas acabado el programa puedes insertar líneas si lo necesitas, utilizando los números 15, 25, 36, etc. El ordenador ejecuta el programa empezando por la que tiene el menor número y siguiendo con las demás en orden creciente.

Antes de escribir un programa, introduce:

NEW <RETURN>

Este comando borra de la memoria del ordenador cualquier programa y sus variables. Te recomendamos que utilices este comando antes de de escribir un nuevo programa.

Ahora escribe el siguiente programa. No olvides pulsar la tecla <RETURN> después de cada línea:

```
10 PRINT "¿CUAL ES TU NOMBRE?"                <RETURN>
20 INPUT N$                                   <RETURN>
30 PRINT "HOLA ";N$;" BIENVENIDO AL MSX"      <RETURN>
```

Ejecuta este programa mediante RUN. Se visualizará lo siguiente, siempre que no hayas cometido ningún error:

¿CUAL ES TU NOMBRE?
?

La instrucción INPUT, en la línea 20, espera que introduzcas un dato de tipo cadena de caracteres. Escribe tu nombre y pulsa <RETURN>.

El ordenador imprimirá:

HOLA nombre BIENVENIDO AL MSX

Después de escribir tu nombre el ordenador continúa la ejecución en la línea 30, visualizando el mensaje de bienvenida. Cuando el programa no tiene más líneas para ejecutar, el ordenador reconoce que ha llegado el fin de éste y visualiza "OK". En este momento te encuentras de nuevo en el modo comando.

Si has obtenido un mensaje de error, en lugar del mensaje de bienvenida, es que has cometido algún error. Antes de buscar los errores lista el programa de nuevo.

Para hacerlo escribe:

LIST

Este comando listará tu programa empezando por el número de línea menor. Todas las palabras reservadas y las variables se listarán en mayúsculas, aunque tú las hayas escrito en minúsculas. Verás:

```
10 PRINT "¿CUAL ES TU NOMBRE?"
20 INPUT N$
30 PRINT "HOLA ";N$;" BIENVENIDO AL MSX"
```

Busca en la línea a que se refiere el mensaje de error. Una vez que hayas encontrado el error, edita esa línea utilizando las teclas de edición y el cursor, tal como describimos en el tema 1. A continuación, con el cursor todavía en la línea ya corregida, pulsa <RETURN>. El ordenador cambiará la línea antigua por la versión correcta.

Antes de ejecutar un programa es conveniente que el cursor se encuentre fuera de éste. Para hacerlo mantén pulsada la tecla <↓> hasta que el cursor se encuentre situado en la zona inferior de la pantalla. Escribe de nuevo RUN seguido de <RETURN>. Repite el proceso de edición hasta que tu programa funcione correctamente.

Si tuvieses toda la pantalla desordenada pulsa <SHIFT><CLS> antes de escribir LIST.

Si has cometido muchos errores en una línea, en vez de editarla y corregirla es más rápido escribirla de nuevo:

10 PRONT CAal os tus notbre!"

Sería más rápido reescribir la línea entera:

10 PRINT"Hola, cuál es tu nombre?"

La antigua línea 10 es sustituida por la nueva.

Si lo que deseas es borrar del programa toda una línea, escribe el número de línea y pulsa <RETURN>. Cuando hayas hecho esto no podrás recuperar la línea que has borrado.

10 <RETURN>

borra la línea 10.

Escribe el programa siguiente:

```
10 REM Este programa se autorrepite
20 CLS
30 INPUT "¿CUAL ES TU NOMBRE";N$
40 PRINT"HOLA ";N$;" BIENVENIDO AL MSX"
50 GOTO 30
```

La sentencia REM es un comentario; el ordenador no tiene en cuenta el contenido de aquellas líneas que lleven la sentencia REM. Esta sentencia se suele

emplear para comentar lo que hace el programa. Es interesante incluir estas instrucciones en tu programa, ya que así no olvidarás qué es lo que hace.

La segunda línea limpia la pantalla. Ya conoces una forma de hacerlo, que es utilizando las teclas <SHIFT> <HOME>, pero si deseas que lo haga tu programa debes utilizar la instrucción CLS.

El comando de la línea 30 ya es familiar. Este ejemplo te muestra cómo imprimir un mensaje antes de que aparezca la "?". Lo que tienes que hacer es poner primero el mensaje entre comillas; a continuación pon un punto y coma, y después el nombre de la variable. El orden aquí es muy importante.

La línea 50 sólo indica que se vuelva a la línea 30, y que desde allí continúe la ejecución. A esto se le llama "bucle infinito", ya que cada vez que la ejecución del programa llegue a la línea 50 ésta saltará a la 30. Continuará hasta que detengas la ejecución.

Prueba a ejecutar el programa.

Cuando te canses de meter tu nombre pulsa <CTRL><STOP>. Obtendrás el siguiente mensaje:

```
BREAK at LINE 30
```

Para continuar ejecutando escribe CONT. La ejecución continuará en la misma línea donde se había cortado, es decir, en la instrucción INPUT de la línea 30.

Si detuviste el programa en cualquier otra línea, es decir, una que no contiene la instrucción INPUT, la ejecución empezará en la siguiente línea a la que se detuvo.

jercicio

1. Escribe un programa que te pregunte el nombre y edad, y lo escriba a continuación.

4

Algo más de aritmética

Ya conoces los símbolos aritméticos:

```
+ - / *
```

Otro símbolo muy útil es "^". Este eleva un número a una potencia; es decir, 2^3 se escribirá 2^3.
Escribe:

```
PRINT 4^5
```

Obtendrás el valor 1.024.
En las expresiones aritméticas puedes utilizar paréntesis. El ordenador evaluará primero las expresiones entre paréntesis.
Puedes combinar en una expresión todos los símbolos indicados anteriormente. La expresión se evaluará en el orden siguiente:

()	Primero se evalúan las expresiones entre paréntesis.
^	Exponenciales.
*,/	La multiplicación y la división tienen la misma prioridad.
+,-	Se evalúan en último lugar.

En el siguiente ejemplo el ordenador evalúa la expresión en tres etapas:

```
                (3 + 4) * 7^2
Primera etapa     7 * 7^2
Segunda etapa     7 * 49
Tercera etapa       343
```

Debes incluir el signo "*" entre los paréntesis; es decir:

PRINT (3 + 4)(4 − 2) ES INCORRECTO

Esta línea se debe escribir:

PRINT (3 + 4) * (4 − 2)

En el siguiente ejemplo se pregunta por una temperatura en grados Fahrenheit y se convierte a centígrados:

```
10 REM Conversion de temperaturas
20 CLS
30 INPUT "TEMPERATURA FARENHEIT";F
40 C=(F-32)*5/9
50 PRINT"TEMPERATURA EQUIVALENTE EN CENTIGRADOS: ";C
60 GOTO 30
```

Advierte que en la línea 40 se ha omitido la sentencia LET. Esta sentencia es opcional. Cuando el ordenador encuentra una variable al lado del signo igual, asigna el valor de la expresión a la derecha del "=" a la variable numérica que se encuentre a la izquierda.

Cuando ejecutas el programa encontrarás que la temperatura en grados centígrados se da con cifras decimales. Si deseas que la temperatura se redondee al entero más próximo utiliza la función INT.

INT convierte números reales, es decir, números que tienen dígitos decimales, a números enteros. El argumento debe ir entre paréntesis.

La función INT redondea un número real al entero menor más próximo.

```
 3.4     lo convierte en   3
−2.7     lo convierte en  −3
```

Para redondear un número real al entero más alto, y no al más bajo, se le debe añadir 0.5:

```
INT(3.9)          se evalúa como 3
INT(3.9 + 0.5)    se evalúa como 4
```

Debes indicar si deseas almacenar números enteros o reales. Para distinguirlos, los números enteros acaban con el signo %.

Sustituye las líneas 40 y 50 del último programa por:

```
40 C%=INT((F-32)*5/9+.5)
50 PRINT "TEMPERATURA EQUIVALENTE EN CENTIGRA
DOS : ";C%
```

El programa también funcionará si utilizas C en vez de C%, pero el número se seguirá almacenando con decimales, incluso cuando todos los números que hay después del punto decimal sean ceros. Los ceros no se escriben, pero se almacenan en memoria y ocupan espacio. Es mejor utilizar, siempre que sea posible, nombres de variables enteras.

Ejercicio

1. Cambia el programa de la temperatura para introducir la temperatura en grados centígrados y obtener su equivalente en Fahrenheit. Te hará falta utilizar la siguiente ecuación:
   F = C * 9/5 + 32

5

Cómo emplear los bucles

Este programa escribe la tabla de multiplicación que deseas. ¡No te aburras escribiéndolo!:

```
10   REM Tabla de multiplicacion
20   INPUT "Multiplicador";M
30   PRINT "una vez";M;"=";1*M
40   PRINT "dos veces";M;"=";2*M
50   PRINT "tres veces";M;"=";3*M
```

y así sucesivamente. Es un programa largo y repetitivo. Una forma mucho más sencilla para hacerlo es utilizando el bucle FOR...NEXT:

```
10   REM Tabla de multiplicacion con bucle
20   INPUT "Multiplicador";M%
30   FOR C%=1 TO 10
40   PRINT C%;"veces";M%;"=";C%*M%
50   NEXT C%
60   GOTO 20
```

En este programa se han utilizado las variables numéricas enteras C% y M%.

La primera vez que el ordenador pasa por la línea 30, pone C% = 1 y pasa a la línea 40. Cuando llega a la línea 50, con la sentencia NEXT vuelve a la 30. En esta línea se incrementa el contador en una unidad, es decir, que ahora C% = 2. Este bucle se repite hasta que C% = 10, que es el valor límite; cuando se llega a este valor, el ordenador ignora la sentencia NEXT de la línea 50 y continúa en la 60.

Puedes hacer que el contador se incremente en más de una unidad, utilizando la sentencia STEP en la línea FOR...NEXT.

Intenta lo siguiente:

```
10 FOR N=0 TO 12 STEP 2
20 PRINT N
30 NEXT
```

Obtendrás

```
0
2
4
6
8
10
12
```

La construcción de los nombres del contador sigue las mismas reglas que las utilizadas para los nombres de variables numéricas. Véase el capítulo 2.

Si en el ejemplo anterior sustituyes el límite superior por 13, los resultados del programa no variarán, ya que empezando por 0 y con intervalos crecientes de dos en dos nunca se llegará al valor 13, por lo que se saldrá del bucle cuando el contador sea igual a 12.

Puedes utilizar tanto números reales como enteros para los valores inicial y final del contador y del intervalo. También se pueden utilizar valores negativos para el intervalo:

```
10 FOR N=10 TO 2.5 STEP -2.5
20 PRINT N
30 NEXT N
```

te dará:

```
10
7.5
5
2.5
```

Cuando el intervalo que utilizas es negativo, debes tener en cuenta que el valor inicial tiene que ser mayor que el final; en caso contrario se producirá un error.

Se puede poner un bucle FOR...NEXT dentro de otro. A esto se le denomina "bucle anidado". Utilizaremos esta técnica en el siguiente ejemplo, para obtener una figura con "*".

```
10 REM Bucle anidado
20 PRINT "*"
30 FOR I=1 TO 9
40 FOR J=1 TO I          'EL
50 PRINT "*";            'BUCLE
60 NEXT J                'ANIDADO
70 PRINT "*"
80 NEXT I
```

Verás:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

El ";" al final de la instrucción PRINT en la línea 50 provoca que la siguiente instrucción PRINT escriba a continuación, en la misma línea, empezando por la primera posición que encuentre libre; es decir, no hay salto de línea.

En la línea 30 pone I = 1, que establece el límite superior para el contador J del bucle anidado, por lo que en la primera vuelta sólo se escribirá un "*". En la siguiente vuelta se incrementa I, asignándose el valor 2. J puede tomar ahora los valores 1 y 2, por lo que esta vez se ejecuta dos veces el bucle, escribiéndose dos "*" en la misma línea.

Variando el tamaño de STEP podrás obtener diferentes figuras.

Ejercicios

1. Mira lo que ocurre si omites las líneas 20 y 70.
2. Intenta modificar el bucle principal del programa anterior añadiéndole STEP 4. Cambia el límite de este bucle. Con esto conseguirás que las líneas 50 y 70 modifiquen la figura.
3. Varía el tamaño del STEP del bucle anidado.

6
Las condiciones

Si necesitas que tu programa tome decisiones dependiendo de unas ciertas condiciones utiliza el formato IF...THEN. Aquí tienes un ejemplo:

```
20 IF A=B THEN GOTO 100
30 PRINT A,B
```

En la línea 20 se le dice al ordenador que si la variable A es igual a la B, salte a la línea 100; en caso contrario son distintos y la ejecución continuará en la siguiente línea, que en este caso es escribir los valores de las variables A y B.

Hay muchas expresiones que pueden ir a continuación de la sentencia IF y pueden utilizar los siguientes signos:

>	MAYOR QUE
<	MENOR QUE
=	IGUAL A
>=	MAYOR O IGUAL QUE
<=	MENOR O IGUAL QUE
<>	DISTINTO QUE

A THEN le puede seguir cualquier comando. Por ejemplo:

```
20  IF A = B THEN PRINT A
```

En el programa siguiente se introducen dos números y se escribe primero el mayor, y luego el menor:

```
10 INPUT "Numeros";A,B
20 IF A>B THEN GOTO 40
30 SWAP A,B
40 PRINT "El numero mayor es:";A
50 PRINT "El numero menor es:";B
```

En este programa se introducen dos números utilizando una única instrucción INPUT. El ordenador visualiza "Números?" y espera hasta que se hayan introducido los dos valores. Tienes dos opciones para meter los números: la primera es escribir el primero, después una coma y a continuación el segundo, y luego pulsar <RETURN>; y una segunda opción es la de escribir el primer número, pulsar <RETURN>, escribir el segundo y pulsar <RETURN>. Si eliges esta segunda opción el ordenador visualizará dos interrogaciones después del primer <RETURN>. Cuando hayas metido los datos se continuará con el programa normalmente.

Se puede poner cualquier número de variables en una única instrucción INPUT, separando las variables entre sí mediante comas y éstas del mensaje entre comillas por un punto y coma.

El comando SWAP de la línea 30 intercambia los valores de las variables, poniendo el contenido de la variable A en la B, y viceversa; ya que si la condición A > B es falsa, en vez de saltar a a la línea 40 se continuará en la 30. El comando SWAP asigna a la variable A el mayor número. Las dos líneas siguientes imprimen los números. También se utiliza SWAP para intercambiar variables de tipo cadena de caracteres. Las variables deben estar separadas por comas.

El formato IF...THEN se puede ampliar con la instrucción ELSE. Se ejecuta lo que sigue a ELSE cuando la condición del IF...THEN es falsa. El programa anterior también se puede escribir como:

```
10 INPUT "Numeros";A,B
20 IF A>B THEN PRINT "El numero mayor es:";A
ELSE PRINT "El numero mayor es:";B
30 IF A>B THEN PRINT "El numero menor es:";B
ELSE PRINT "El numero menor es:";A
```

Se puede utilizar más de una condición en la sentencia IF...THEN utilizando AND y OR. Por ejemplo:

```
10 INPUT A,B,C
20 IF A=B AND A>C THEN PRINT A
```

Este pequeño programa sólo imprimirá A, siempre y cuando A sea igual a B y mayor que C. Compara el anterior ejemplo con el siguiente:

```
10 INPUT A,B,C
20 IF A=B OR A>C THEN PRINT A
```

Esta versión del programa anterior imprimirá A cuando sea igual a B o mayor que C.

Con cadenas de caracteres también se pueden utilizar desigualdades, ya que éstas se encuentran almacenadas en memoria como número; cada carácter se encuentra representado por un número. Para comparar dos cadenas de caracteres el ordenador compara los números representativos del primer carácter de cada cadena. Si los dos primeros son iguales se pasa a comparar los dos segundos, y así sucesivamente. Las letras mayúsculas y minúsculas se representan con diferentes números:

De la A a la Z por los números 54 a 90.
De la a a la z por los números 97 a 122.

También se pueden comparar otro tipo de caracteres como "*". Pero hablaremos más tarde. (Véase el tema 20.)

Las siguientes cadenas de caracteres se ordenan de mayor a menor según el número correspondiente:

"cebra"
"aa"
"aA"
"a"
"Hola"
"AND"

Para resumir todo lo que has aprendido en este tema, aquí tienes un pequeño programa que utiliza las sentencias IF, THEN y SWAP.

Este programa busca el mayor de tres números dados, visualizándolos y colocando en primer lugar el mayor:

```
10 REM BUSQUEDA
20 INPUT "Tres numeros";A,B,C
30 IF B>A THEN SWAP A,B
40 IF C>B THEN SWAP B,C
50 IF B>A THEN SWAP B,A
60 PRINT A,B,C
```

Ejecuta este programa introduciendo unos cuantos números.

Ejercicios

1. Utilizando el programa anterior, inventa uno que escriba tres cadenas de caracteres en orden descendente.
2. El programa sólo funciona con tres números. ¿Puedes escribir uno que funcione con más números? Tienes uno en el tema 13.

7

Comandos útiles e indicaciones para escribir programas

Un comando muy útil y que ahorra mucho tiempo es AUTO. Cuando lo ejecutes verás que aparece automáticamente el número de línea 10 en la pantalla. Escribe REM y pulsa <RETURN>. Ahora aparecerá el número de línea 20. El proceso continuará indefinidamente. Para parar la numeración automática de líneas pulsa <CTRL><STOP>.

Antes de escribir un nuevo programa deberás utilizar el comando NEW, que borra cualquier programa antiguo e inicializa las variables de éste. Si no lo hicieras podrías encontrar líneas del programa antiguo mezcladas con las del nuevo, especialmente si el antiguo es más largo que éste, con lo que perjudicarás el nuevo programa.

La instrucción LIST se utiliza para listar por pantalla un programa. Si lo que quieres es listar únicamente una línea (por tener un mensaje de error referido a ella), escribe LIST seguido del número de línea:

```
LIST 30
```

Visualiza la línea 30 por pantalla.

Cuando listes tu programa te darás cuenta de que todas las palabras claves y variables que habías escrito en minúsculas te aparecen en mayúsculas.

Titula tu programa mediante la instrucción REM; utilízala también para explicar cada parte de tu programa. Esto facilitará la edición del programa, especialmente si hace tiempo que lo has escrito.

Puedes borrar una sentencia REM si te das cuenta de que estás sobrepasando la capacidad de la memoria. No refieras nunca un comando GOTO a una línea con una sentencia REM, porque si borras dicha línea el programa no se ejecuta.

La manera más sencilla de borrar una línea del programa es escribir el número de línea y pulsar <RETURN>. Si deseas borrar toda una parte del programa es más rápido utilizar el comando DELETE:

```
DELETE 20-80
```

Borrará todas las líneas cuyos números se encuentren entre 20 y 80 inclusive. Ten cuidado de no borrar líneas por accidente, ya que una vez que has usado el comando sólo podrás recuperar esas líneas reescribiéndolas de nuevo.

Se puede incluir más de un comando en una línea, pero siempre deben ir separados por dos puntos. Por ejemplo:

```
20  PRINT "a": GOTO 10
```

Esta línea es válida. Se imprimirá la letra "a" y el ordenador saltará a la línea 10. De esta forma puedes incluir cualquier número de comandos en una línea. Esto ahorra memoria. De cualquier manera, un programa estará mejor redactado si se evita la utilización de sentencias múltiples. En una línea, el número límite de caracteres es de 255, incluyéndose los espacios. El ordenador ignorará todo carácter que sobrepase este límite.

Para comprobarlo escribe lo siguiente:

```
10  PRINT"**************..."
```

Llena la pantalla con "*". A continuación pulsa <SHIFT><CLS> y ejecuta el programa con RUN; sólo se escribirán 247 "*". El ordenador sólo ha ejecutado los doscientos cincuenta y cinco primeros caracteres de una línea. Si te olvidas de las comillas al final de la cadena de caracteres, el ordenador asume automáticamente su presencia. Sin embargo, no tomes este hábito, ya que puede producir resultados confusos.

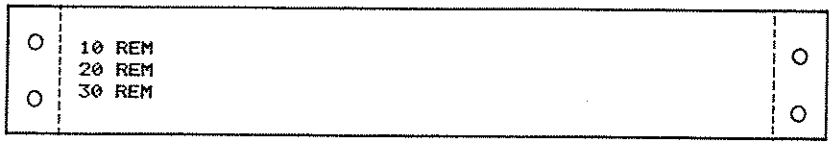
La instrucción PRINT en forma abreviada es el signo de interrogación. Escribe:

```
10?
```

Ahora haz LIST. La interrogación ha sido sustituida por PRINT.

Al revisar tu programa te puedes encontrar con que has olvidado introducir una o varias líneas por accidente cuando lo tecleaste. Es posible que no haya sitio para insertarlas por un espaciamiento inadecuado de las líneas del programa. Si esto ocurre puedes renumerar tu programa utilizando la sentencia RENUM.

Para insertar otra línea entre la 10 y la 11 escribe RENUM. Ahora lista tu programa (LIST) y verás:



Ahora puedes insertar nuevas líneas fácilmente entre la 10 y la 20, utilizando los números de 11 a 19.

Es posible almacenar más de un programa en memoria. Para hacerlo puedes almacenar el primer programa entre los números de línea 10 y 100, y el segundo entre 500 y 1.000. Cuando hagas RUN debes prevenir que el ordenador ejecute el primer programa y luego siga con el segundo. Para evitarlo utiliza la sentencia END, que debe ser la última instrucción del primer programa. Cuando el ordenador se encuentra con END detiene la ejecución y vuelve al modo directo.

Para continuar con la ejecución, o sea, ejecutar el segundo programa almacenado en memoria, escribe CONT seguido de <RETURN>.

Para ejecutar el segundo programa antes que el primero utiliza:

```
RUN 500
```

que empezará a ejecutar a partir de la línea 500. Se obtiene el mismo resultado utilizando:

```
GOTO 500
```

También puedes utilizar el comando STOP dentro de un programa. Este comando, al igual que END, devuelve el control al modo directo. Sin embargo, produce el mensaje "BREAK in LINE...", dando el número de línea de la instrucción STOP. Después de esta instrucción, como te encuentras en modo directo, puedes continuar la ejecución escribiendo CONT.

Puedes parar la ejecución de un programa pulsando <STOP>. Para continuar con la ejecución pulsa <STOP> de nuevo. En este caso no puedes utilizar el comando CONT, ya que su uso sólo está permitido mientras que STOP se encuentre dentro de un programa. Si pulsas <CTRL><STOP>, se para el programa y se vuelve al modo directo. Una vez en el modo directo, se puede continuar la ejecución a partir de la siguiente línea utilizando CONT. Escribe el siguiente programa:

```
10 REM
20 PRINT "Adios"
30 GOTO 20
```

Ejecuta este programa (RUN). Pulsa <STOP> y lo detendrá. La única manera de continuar con la ejecución será pulsando de nuevo la tecla <STOP>. Ejecuta el programa de nuevo, pero esta vez páralo utilizando <CTRL><STOP>. Continúa con él utilizando CONT.

```
10 REM
20 PRINT "HOLA"
30 STOP
40 PRINT "ADIOS"
50 END
60 GOTO 20
```

Continúa con la ejecución del programa utilizando CONT, después de las instrucciones END y STOP.

Un comando distinto, que por el momento no necesitas pero que debes conocer, es CLEAR, el cual inicializa las variables del programa almacenadas en memoria. También se puede utilizar para ampliar el tamaño de una variable. Las cadenas de caracteres sólo pueden tener hasta 200 caracteres con longitud estándar, que podrás ampliar a 255 utilizando:

CLEAR 255

Otro comando relacionado con la memoria del ordenador es FRE, que te indica la cantidad de memoria libre disponible en el área de programas BASIC o en el área de las cadenas de caracteres.

PRINT FRE(0)	Proporciona la memoria disponible en el área de programas BASIC.
PRINT FRE("")	Proporciona la memoria disponible en el área de cadenas de caracteres.

Finalmente, dos comandos que ayudan a descubrir errores en tu programa son: TRON (TRace ON), que hace que el ordenador visualice el número de línea que va a ejecutar.

Escribe:

```
10 REM Uso de TRON
20 PRINT "Hola, ¿aun estas levantado?"
30 INPUT "Si o No";A$
40 IF A$="No" THEN BEEP: GOTO 20
50 PRINT "A la cama"
```


Escribe TRON y ejecuta el programa. Obtendrás lo siguiente:

```
[10][20]  Hola, ¿aún estás levantado?
[30]      ¿Sí o no?
```

Aquí el programa espera una entrada de datos. Si tecleas "Sí" obtendrás:

```
[40][50]  A la cama
```

Si introduces "NO" obtendrás:

```
[40][20]  Hola, ¿aún estás levantado?
[30]      ¿Sí o no?
```

El comando BEEP de la línea 40 es el más sencillo de los utilizados para que el ordenador genere sonido.

Para detener el efecto de TRON utiliza TROFF.

8

Las teclas de función

Son las cinco teclas que se encuentran situadas en la parte superior izquierda del teclado. Cada una tiene dos funciones, una de las cuales se puede utilizar al pulsar la tecla directamente, y la otra pulsando simplemente <SHIFT> y la tecla en cuestión; o sea, las funciones F1, F2, F3, F4 y F5 se obtienen pulsando directamente la tecla correspondiente, mientras que F6, F7, F8, F9 y F10 se obtienen pulsando a la vez <SHIFT> y la tecla correspondiente.

Al conectar el ordenador, las teclas ya se encuentran programadas para determinadas funciones. Se puede visualizar un listado de estas funciones al ejecutar el comando:

KEY LIST

El contenido de las cinco primeras teclas se visualiza en la última línea de la pantalla. Las otras cinco, al pulsar <SHIFT>. Para dejar de visualizar las funciones de las teclas se ha de escribir KEY OFF, y para volver a visualizarlas en pantalla utiliza KEY ON. Si lo deseas, puedes incluir los tres últimos comandos en un programa.

Por el momento olvídate de las teclas F1, F6 y F7. Se hablará de ellas más adelante. Las funciones de las otras siete teclas te serán ya familiares y probablemente útiles:

Tecla de función	Se visualiza	Descripción
F1	color	Véase el tema 22.
F2	auto	Está programado para visualizar AUTO por pantalla, seguido de un espacio, por lo que esta tecla es equivalente a escribir "AUTO".
F3	goto	Pulsar esta tecla es equivalente a escribir GOTO seguido de un espacio.
F4	list	Es equivalente a escribir LIST seguido de un espacio.
F5	run	La función de esta tecla es algo diferente a las anteriores. Es equivalente a escribir RUN seguido de <RETURN>, por lo que al pulsar esta tecla se ejecutará cualquier programa almacenado en memoria.
F6	color 15,4,7	Véase el tema 22.
F7	cload"	Véase el tema 11.
F8	cont	Es equivalente a CONT seguido de <RETURN>. Si has detenido la ejecución de un programa mediante <CTRL><STOP>, o utilizando las instrucciones END o STOP, podrás continuar la ejecución a partir de la siguiente línea pulsando <SHIFT> y F8.
F9	list	F9 lista el programa desde la última línea introducida, y sitúa el cursor al principio de esa línea.
F10	run	Esta función es útil, ya que limpia la pantalla y a continuación ejecuta cualquier programa que se encuentre en memoria.

Es posible redefinir las funciones de estas teclas utilizando el comando KEY, seguido por un número de tecla, una coma y la cadena de caracteres equivalente a la nueva función.

Escribe:

```
KEY 1,"PRINT"  <RETURN>
```

Nos visualizará la instrucción PRINT por pantalla cada vez que pulsemos la tecla de función F1.

Date cuenta que la lista que aparece en la parte inferior de la pantalla ha cambiado; COLOR ha sido sustituido por PRINT.

```
KEY 6,"NEW" + CHR$(13)
```

Programa la tecla 6 para que utilice el comando NEW. CHR$(13) es equivalente a pulsar <RETURN>. (Para más detalles véase el tema 20.)

Cuando definas una tecla no podrás utilizar en la cadena de definición más de quince caracteres. Recuerda que CHR$(13) sólo ocupa la posición de un carácter.

Ejercicio

1. Programa la tecla de función F7 para que al pulsar esa tecla se renumere tu programa.

9

Algo más sobre las salidas (PRINT) y la pantalla

Ya has utilizado la instrucción PRINT para escribir cadenas de caracteres y variables, separando la lista de los elementos tanto por una coma como por un punto y coma. El punto y coma escribe los elementos uno a continuación del otro; la coma escribe dos elementos en la misma línea, el primero al principio de la línea y el segundo en la posición 15.

El punto y coma al final de una instrucción PRINT evita el salto de línea de la siguiente instrucción PRINT:

```
10 PRINT "Ser o no ser -";
20 PRINT "He ahi el dilema;"
```

Al ejecutar estas dos líneas, los dos PRINT escribirán en la misma línea:

Ser o no ser - He ahí el dilema;

Probablemente te gustaría poder escribir una tabla de resultados o una lista de opciones totalmente tabuladas. Tienes dos comandos que te ayudarán a hacerlo: TAB y LOCATE.

Para entender la utilización de estos comandos, primero debes conocer los distintos modos de pantalla.

Hay dos pantallas de texto: SCREEN 0 y SCREEN 1. Cuando conectas el

ordenador entras automáticamente en una de esas pantallas. En este tema asumimos que te encuentras en SCREEN 1. Para estar seguro de ello escribe:

```
SCREEN 1
```

Esta pantalla tiene 29 caracteres de ancho por 24 de largo. Cada posición de un carácter ocupa un cuadrado de 8 × 8 puntos.

La otra pantalla de texto, SCREEN 0, tiene 39 caracteres de anchura por 24 de largo. El número de puntos por carácter, en esta pantalla, es menor, ya que cada uno ocupa un rectángulo de 6 × 8 puntos. En esta pantalla el borde es del mismo color que el fondo, por lo que toda la pantalla es de color azul cuando se conecta el ordenador. Para acceder a esta pantalla escribe:

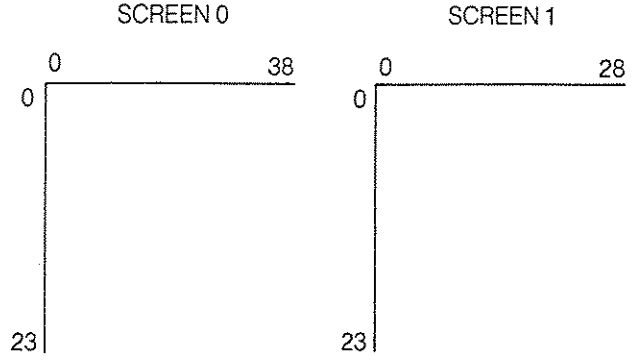
```
SCREEN 0
```

Ahora vuelve a la pantalla 1, escribiendo:

```
SCREEN 1
```

El origen de coordenadas de ambas pantallas de texto se encuentra situado en la parte superior izquierda de la pantalla. Observa que a la primera fila y columna de cada pantalla se la llama fila 0 y columna 0, respectivamente. Por lo que la última columna de la pantalla 0 es la columna 38, y en la pantalla 1 es la columna 28. La última fila en ambos casos es la fila 23.

No puedes utilizar la última fila a menos que utilices la sentencia KEY OFF para borrar el menú de las teclas de función.



Puedes cambiar el máximo número de caracteres por línea utilizando la sentencia WIDTH, seguida de un número. Este número establece la máxima anchura de la pantalla. En la pantalla 0, la máxima anchura que puedes tener es de 40 caracteres, y en la pantalla 1 es de 32. Puedes decrementar la anchura de ambas pantallas hasta un carácter por línea.


```
SCREEN 0
WIDTH 20
```

Este comando ha fijado la máxima anchura de la pantalla 0 en 20 caracteres. Ahora sólo podrás escribir en las veinte columnas centrales.

Para volver al tamaño original escribe:

```
SCREEN 0
WIDTH 39
```

Ahora pasamos a la tabulación. El comando TAB siempre se ha de utilizar en conjunción con PRINT. Esta instrucción hace que el ordenador comience a escribir en la columna especificada (o coordenada x) dada por la sentencia TAB. Por ejemplo:

```
PRINT TAB(10)"Empieza a escribir en la posición del undécimo carácter"
```

No olvides que la posición del primer carácter de la pantalla se encuentra en la columna 0, fila 0. Puedes pensar que es una posición de coordenadas X,Y (0,0).

Es posible utilizar más de una instrucción TAB en una sentencia PRINT:

```
PRINT TAB(3)4 TAB(7)8
```

que escribirá el número 4 en la cuarta columna. El espacio en blanco que siempre precede a un número positivo se escribe en la tercera columna. El número 8 se escribirá en la columna 8 de la misma línea precedido por un espacio en blanco en la columna 7. No se necesita ninguna puntuación entre las instrucciones TAB, pero no te olvides de los paréntesis.

En la instrucción TAB sólo puedes especificar la columna; pero si lo que deseas es especificar una determinada columna y una fila, utiliza el comando LOCATE justo antes del comando PRINT. Este comando fija la posición del cursor; el primer número del comando LOCATE es el de la columna (coordenada X), el segundo es el de la fila (coordenada Y). Por ejemplo:

```
10 LOCATE 6,2
20 PRINT "NOMBRE"
30 LOCATE 20,2
40 PRINT "APELLIDO"
```

que escribirá:

```
NOMBRE   APELLIDO
```

en la tercera línea de la pantalla.

Para cada sentencia PRINT se necesita un comando LOCATE.

También puedes utilizar LOCATE para fijar únicamente las filas, y entonces utilizar también PRINT TAB. El siguiente programa proporciona el mismo resultado que el anterior.

```
10 CLS
20 LOCATE ,2
30 PRINT TAB(6)"NOMBRE" TAB(20)"APELLIDO"
```

Cuando en la sentencia LOCATE no se incluye coordenada X, el ordenador asume que deseas continuar con la posición actual de X. El comando CLS de la línea 10 ha fijado X = 0, por lo que utiliza este valor de X cuando localiza el cursor en la línea 20. Observa que, aunque no se incluye la coordenada X, se debe escribir la coma.

LOCATE no se puede utilizar en el modo comando, ya que tan pronto como se ejecute un comando en este modo el cursor se fija a la izquierda de la siguiente línea y se prepara para que se introduzca un nuevo comando.

También puedes utilizar LOCATE para hacer desaparecer al cursor de texto, es decir, para no visualizar ese pequeño cuadradito blanco. Para hacerlo pon una coma después de la coordenada Y seguida de un cero. Para visualizar el cursor otra vez sustituye el 0 por un 1. Por ejemplo:

```
10 LOCATE ,,0
```

borra el cursor. Mientras que:

```
10 LOCATE 10,10,1
```

habilita el cursor y lo mueve a la posición de coordenadas (10,10).

Otro comando es el SPC, que se utiliza con la sentencia PRINT para especificar el número de espacios que se desean escribir. El programa anterior, que escribía "NOMBRE" y "APELLIDO", puede ser sustituido por:

```
10 CLS
20 LOCATE ,2
30 PRINT TAB(6)"NOMBRE"SPC(8)"APELLIDO"
```

En una instrucción SPC se puede poner cualquier número que se encuentre comprendido entre 1 y 255, pero debe estar entre paréntesis. Cuando desees escribir una cadena de caracteres que contenga muchos espacios es aconsejable utilizar la sentencia SPC, ya que ahorra mucha memoria.

Hay otro método alternativo para escribir espacios, utilizando SPACE$. Este

es mucho más versátil, ya que también puede utilizarse fuera de la sentencia PRINT, cosa que no ocurría con SPC. SPACE$ se utiliza para formar cadenas de caracteres que sólo contengan espacios en blanco. Por ejemplo:

```
A$ = SPACE$(10)
```

es equivalente a:

```
A$ = "          "
```

Utilizando este comando podemos asignar una determinada cabecera a una variable y escribirla tan a menudo como deseemos:

```
10 H$="NOMBRE"+SPACE$(8)+"APELLIDO"
20 CLS
30 LOCATE ,2
40 PRINT TAB(6)H$
```

No te olvides que esto no se puede hacer con SPC:

```
A = SPC(10)  ¡ES INCORRECTO!
```

Ejercicio

1. Escribe un programa que tenga como entradas la fecha de hoy y el día de tu cumpleaños y que te dé como salida tu edad. Tabula los mensajes que salgan en pantalla con LOCATE. Haz desaparecer el cursor antes de ejecutar la sentencia INPUT.

10

Programación interactiva

A un programa se le llama "interactivo" cuando en algún punto de su ejecución acepta una entrada desde el teclado. En este tema se hablará de las sentencias INPUT, INKEY$ e INPUT$.

Aunque TAB sólo puede utilizarse con sentencias PRINT, LOCATE también se puede utilizar con la instrucción INPUT:

```
10 LOCATE 10,22
20 INPUT "Articulo,Precio";A$,P
```

Verás:

Artículo,Precio?

que aparecen en la fila 22 de la pantalla. El ordenador esperará hasta que hayas introducido una cadena y un número. Hazlo como ya sabes: puedes pulsar <RETURN> después de introducir la cadena de caracteres y después introducir el número, o introducirlos en la misma línea separados por una coma. Sin embargo, si utilizas la sentencia LINE INPUT podrías introducir la línea entera del texto en una única variable de tipo cadena de caracteres:

```
10 LOCATE 10,22
20 LINE INPUT "¿Articulos,Precio?";L$
```

Verás:

¿Artículos,Precio?

en la fila 22 de la pantalla. Ahora, si escribes:

Plátanos, melocotones y azúcar 25.00

la línea entera, incluyendo los blancos, se le asignará a la variable L$. Si sustituyes LINE INPUT por una sentencia INPUT, se ignora todo lo que haya después de la coma.

LINE INPUT sólo se puede utilizar con variables de tipo cadena de caracteres, por lo que:

```
10 LINE INPUT A
```
ES INCORRECTO

La línea que escribas puede tener una longitud máxima de 200 caracteres, pero se puede ampliar a 255 utilizando la sentencia CLEAR. (Véase el tema 7.)

A diferencia de INPUT, LINE INPUT no escribe un signo de interrogación de forma automática, por cuanto si lo deseas debes incluirlo en el mensaje, después de la instrucción. Se puede introducir más de una línea en una sentencia LINE INPUT, pero, entonces, las variables de tipo cadena de caracteres deben separarse unas de las otras por comas, y éstas del mensaje por punto y coma.

Puedes desear que en algún punto el ordenador se pare y te pregunte entre varias opciones a elegir. Por ejemplo, en un programa de juego querrás que el ordenador te pregunte si deseas que se visualicen las instrucciones o, al final del juego, si deseas volver a jugar. Puedes utilizar la instrucción INKEY$ para hacerlo. Esta función analiza si se ha pulsado alguna tecla. Si así fuese, se asigna el carácter de la tecla pulsada a la variable de tipo cadena de caracteres INKEY$. En caso contrario devuelve una cadena de caracteres vacía y el ordenador continuará con la siguiente línea.

Puedes incluir la siguiente secuencia al final de un programa de juegos:

```
100 CLS
110 LOCATE ,10
120 PRINT "¿Quieres seguir jugando?"
130 PRINT "Pulsa S (SI)"
140 PRINT "Pulsa N (NO)"
150 A$=INKEY$
160 IF A$="N" OR A$="n" THEN END
170 IF A$="S" OR A$="s" GOTO 10
180 GOTO 150
```


La primera vez que se pasa por la línea 150, INKEY$ puede contener una cadena de caracteres vacía o bien cualquier carácter que se haya pulsado. Si se ha pulsado una de las teclas N, n, S o s, mientras que el ordenador aún estaba en la línea 150, entonces el programa finaliza o vuelve a empezar desde la línea 10. La instrucción GOTO de la línea 180 hace que la ejecución salte a la línea 150 hasta que se haya pulsado alguna tecla de las pedidas.

No olvides que INKEY$ es una cadena de caracteres y sólo puede asignarse a una variable de tipo cadena de caracteres; por tanto,

A = INKEY$ ES INCORRECTO

y te producirá como resultado un mensaje de error.

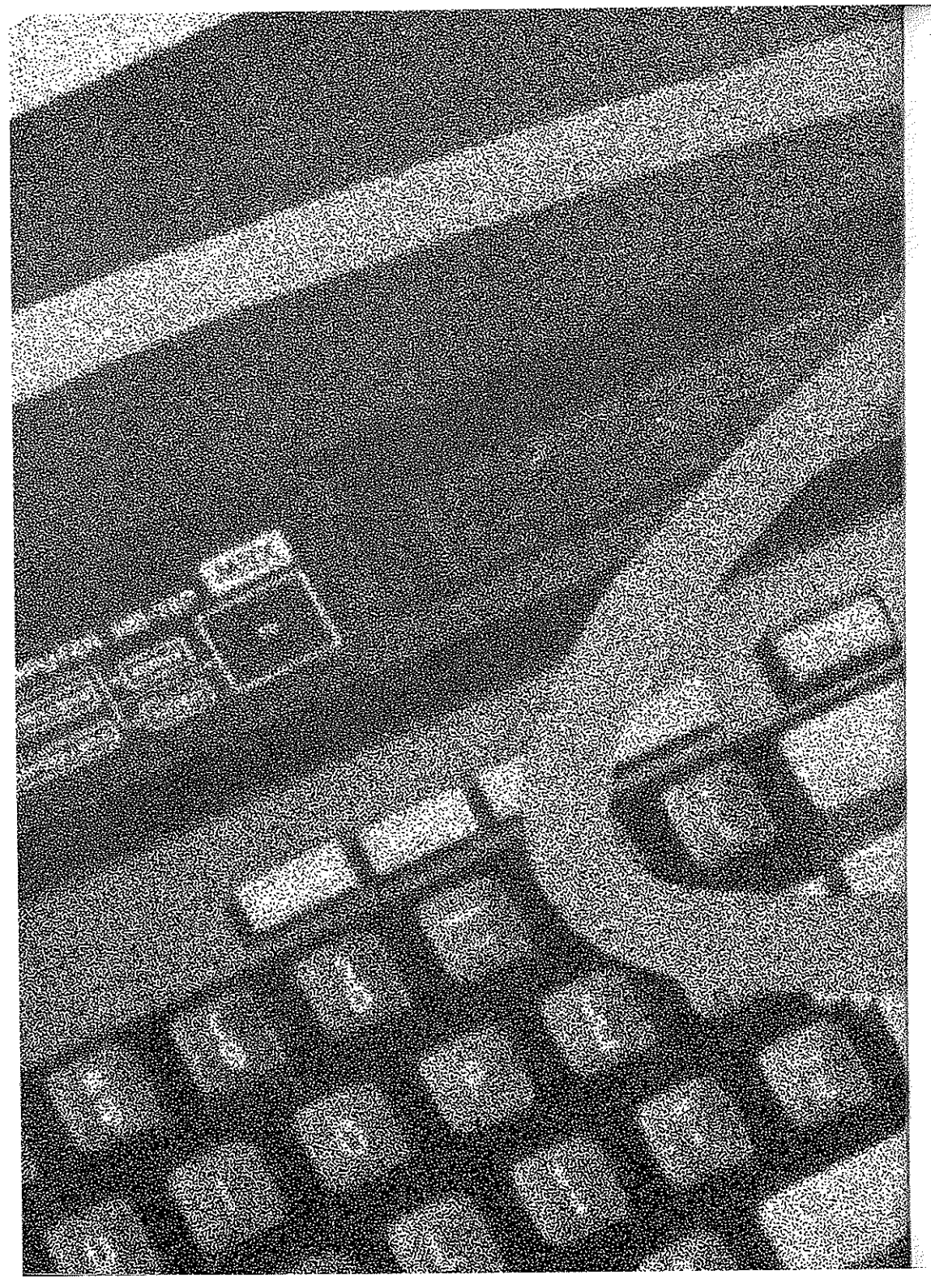
Una instrucción similar a INKEY$ es INPUT$. Al igual que INKEY$, ésta lee directamente del teclado, pero en vez de leer un carácter lee un determinado número de caracteres, especificado por la instrucción INPUT$, es decir:

```
10 A$=INPUT$(5)
20 PRINT A$
```

El ordenador esperará en la línea 20 hasta que se hayan pulsado cinco teclas. La línea 20 escribe esos caracteres. El número del argumento asignado a INPUT$ debe estar comprendido entre 1 y 200 y debe ser entero. Si se utilizase un número real dentro de ese rango, se truncará al entero inferior más próximo.

Ejercicio

1. Utiliza la sentencia INPUT$ al principio de un programa con el objeto de introducir la palabra clave. Las primeras líneas del programa deberán preguntar por ella y la ejecución del resto del programa sólo deberá continuar cuando se haya introducido correcta.



11

Grabación de programas en cassette

Ha llegado el momento en el que puedes escribir programas bastante largos, y en lugar de escribirlos una y otra vez encontrarás más conveniente almacenarlos en un cassette.

Primero conecta tu cassette al ordenador. La mayoría de ellos sirven. Es útil que tenga un contador de vueltas, para poder anotar en dónde has almacenado tu programa. El cassette debe tener una entrada para conectar el micrófono y una salida para conectar auriculares.

El cable de conexión tiene un enchufe con ocho clavijas DIN en un extremo. Si el cable no viene incluido con tu ordenador MSX, entonces cómprate uno. Debes especificar que es para un MSX. Conéctalo al ordenador en el enchufe que pone "CASSETTE". El otro extremo del cable se encuentra dividido en tres conexiones, que tiene sus extremos de color rojo, negro y blanco.

Pon el blanco en el enchufe marcado con "EAR" en el cassette, y el rojo en el que pone "MIC". Olvídate del negro por el momento.

Pon una cinta virgen en tu cassette. Rebobínala si es necesario poniendo el cassette en marcha hasta que llegues al principio de la cinta. Pon a cero el contador de vueltas.

Ahora escribe un programa corto en el ordenador. A continuación escribe:

```
CSAVE"nombre"  <RETURN>
```

Puedes darle a tu programa el nombre que desees con tal de que éste no

exceda de seis caracteres. Los caracteres pueden ser letras mayúsculas o minúsculas o números. El primer carácter del nombre debe ser una letra.

Ahora pulsa los botones RECORD y PLAY de tu cassette y luego <RETURN> en tu ordenador.

Cuando el ordenador acabe de almacenar el programa en la cinta se verá el mensaje "OK" en la pantalla. Detén el cassette y anota el número del contador.

Para saber si has grabado el programa satisfactoriamente desconecta el ordenador para que se borre el programa de la memoria. Rebobina la cinta hasta que el contador esté a cero, vuelve a conectar el ordenador y escribe:

```
CLOAD"nombre"  <RETURN>
```

o simplemente:

```
CLOAD  <RETURN>
```

Si no se especifica el nombre del programa, el ordenador cargará el primero que encuentre en la cinta. Ahora pulsa el botón PLAY del cassette. Obtendrás el siguiente mensaje por pantalla cuando el ordenador haya encontrado al programa:

```
FOUND:nombre
```

Cuando se haya cargado el programa en memoria verás el mensaje "OK"; cuando ocurra esto para el cassette.

Recuerda que la tecla de función F7 está programada para visualizar por pantalla CLOAD". Si utilizas esto sólo necesitarás escribir el nombre del programa y las comillas finales.

Se puede verificar si el programa almacenado en memoria y el grabado en cinta son los mismos, es decir, que no se han producido errores al grabar y cargar el programa; para ello emplea el comando CLOAD? Rebobina la cinta hasta que el contador de vueltas esté a cero y escribe:

```
CLOAD?"Nombre"  <RETURN>
```

Pon en funcionamiento el cassette.

Si el programa en memoria y el grabado en cassette son los mismos obtendrás el mensaje "OK". En caso contrario el programa no se ha cargado bien (consulta la sección que habla de estos problemas, al final de este tema).

Cuando grabes varios programas en una misma cinta es conveniente que se encuentren espaciados unos de otros y anotes la lectura del contador de vueltas al principio de cada programa. Si grabas uno encima de otro, obviamente perderás el primero que habías grabado.

Suponiendo que tienes dos programas almacenados en cinta, al primero llámalo "plof" y al segundo "boing". Si haces:

```
CLOAD"boing"
```


y pones en funcionamiento el cassette desde el principio obtendrás lo siguiente:

```
CLOAD"boing"
Skip:plof
Found:boing
OK
```

El ordenador te indica que ha pasado por un programa llamado "plof" y ha cargado "boing".

La clavija negra sólo podrá usarse si tienes una entrada para control remoto en el cassette, marcado por "REMOTE". Si la tienes conecta en él la clavija negra y escribe:

```
MOTOR ON      <RETURN>
```

Si ahora ejecutas CLOAD/CLOAD? o CSAVE, el ordenador desconectará automáticamente el motor del cassette cuando haya acabado de leer o escribir en la cinta. Es muy útil, ya que es fácil olvidarse de desconectar el cassette después de utilizar CLOAD o CSAVE.

Para desactivar el control remoto del cassette escribe:

```
MOTOR OFF  <RETURN>
```

o simplemente:

```
MOTOR  <RETURN>
```

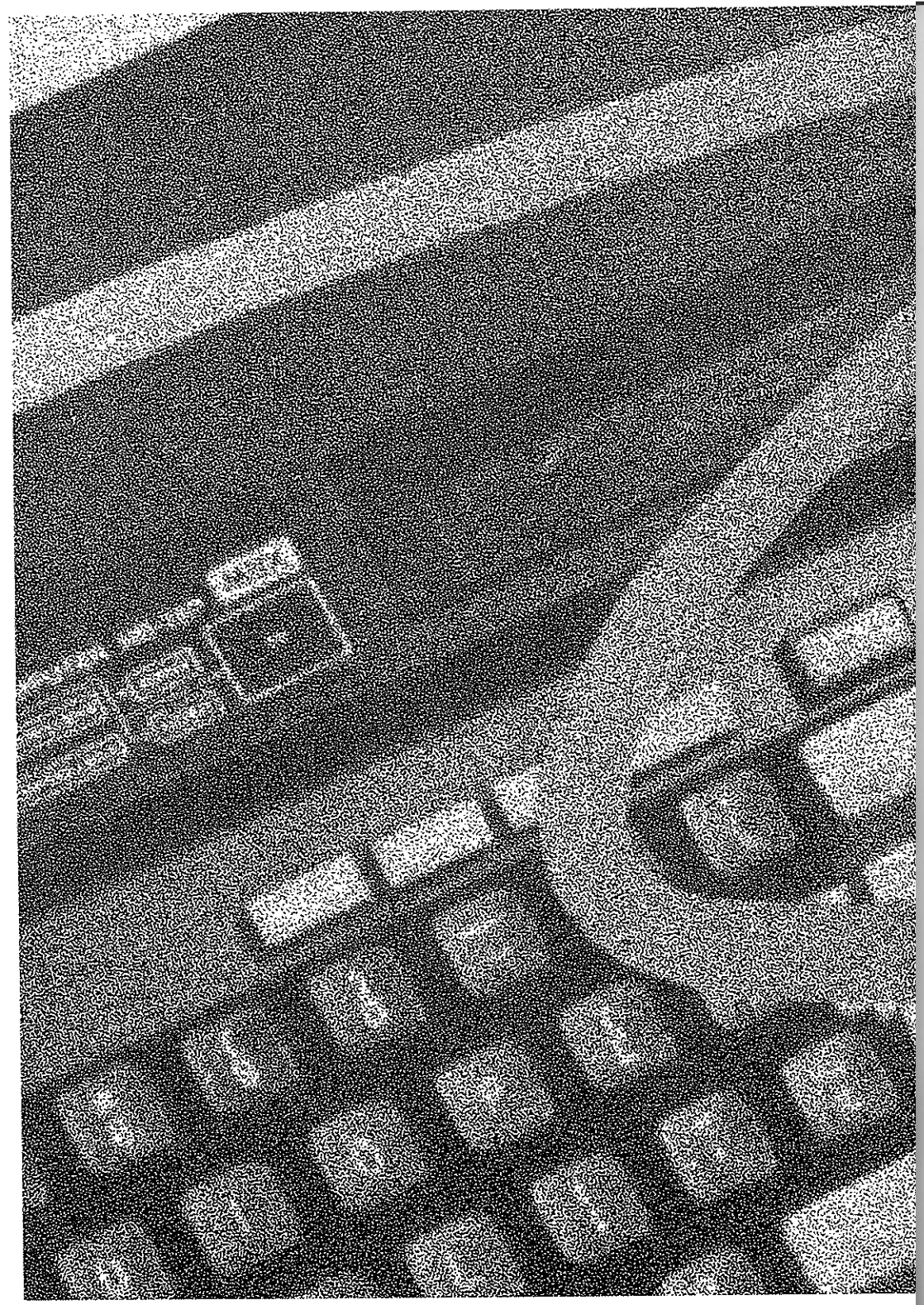
que tiene el mismo efecto, ya que desconecta el motor si estaba conectado y lo conecta en caso contrario.

Errores de carga y grabación

Si has tenido problemas al grabar o cargar tus programas revisa lo siguiente:

1. Si has conectado el cassette.
2. Si has conectado correctamente las clavijas y no se han salido de su sitio.
3. Varía el tono y el volumen de tu cassette. Normalmente es mejor fijar al máximo el tono e ir variando poco a poco el volumen. Empieza con un volumen medio e intenta aumentarlo y disminuirlo a partir de esa cota.
4. Revisa el nombre del programa que has grabado y no puedes cargar.

Si continúas con problemas, probablemente tengas un cassette que no esté en buenas condiciones.



12
Lectura de datos de matrices

Hasta ahora sólo has introducido un número de elementos limitado a la vez. Pero llega el momento en que tienes que almacenar una gran cantidad de datos, posiblemente una larga lista de nombres o números. Supón que quieres tener una guía telefónica de la gente que conoces. Prueba lo siguiente:

```
10 REM GUIA TELFONICA Version 1
20 INPUT "Nombre y Numero";N1$,NUM1
30 INPUT "Nombre y Numero";N2$,NUM2
40 .......
```

y sigue; pero sólo con diez nombres se convertiría en un programa largo y repetitivo. Sería más correcto utilizar un bucle que se ejecutase una vez por cada nombre y nímero que metiese en la guía. Para poder hacerlo necesitas una matriz.

Una matriz es un grupo de variables del mismo tipo, pero que difieren unas de otras en el número del índice.

A(n) es una matriz numérica. Si n = 10, la matriz tendrá once variables llamadas A(0), A(1), A(2), ..., A(10). Imagina que A es el nombre de una calle; en este caso el número que sigue te da el número de la calle.

Si tienes que utilizar matrices de más de once elementos deberás indicarle al ordenador el tamaño y el nombre de la matriz, utilizando la sentencia DIM:

```
DIM NUM(12)
```

Se trata de una matriz numérica de una dimensión llamada NUM y que tiene trece elementos: NUM(0), NUM(1), ..., NUM(12). A la matriz se le llama de una dimensión, ya que puede representarse en un diagrama de una única dimensión:

NUM(0)
NUM(1)
NUM(2)
NUM(3)
NUM(4)
NUM(5)
NUM(6)
NUM(7)
NUM(8)
NUM(9)
NUM(10)
NUM(11)
NUM(12)

El primer carácter del nombre de una matriz debe ser alfabético, el resto pueden ser caracteres numéricos o alfabéticos. Al igual que en las variables simples, no pueden utilizarse en los nombres palabras claves o signos de puntuación. El ordenador sólo reconoce los dos primeros caracteres y no distingue entre letras mayúsculas o minúsculas. Por tanto, las matrices siguientes serán iguales para el ordenador:

ab(10)
AB(10)
Abaco(10)

Si quieres una matriz de cadenas de caracteres, añade el símbolo $ al final del nombre de la matriz. La única diferencia entre ésta y una matriz numérica aparentemente es que el nombre acaba con el signo "$" (al igual que una variable del tipo cadena de caracteres).

Son nombres permitidos:

Juego$(20)
Nombre$(10)
A2$(3)

Los siguientes no están permitidos:

TOWN$(12) Incluye la instrucción TO.
Como?$(6) Los signos de puntuación no están permitidos.
NAME$(200) NAME es una palabra reservada.

Se pueden dimensionar todas las matrices que se van a utilizar en un programa mediante una única instrucción DIM:

```
DIM A$(12),B(13),C(20)
```

No dimensiones matrices con más espacio del que necesitas, ya que al ejecutar la instrucción DIM el ordenador reserva una parte de la memoria para cada matriz, o sea, si no utilizas por completo la matriz estarás desperdiciando memoria.

Se puede ahorrar memoria utilizando matrices de números enteros en vez de matrices de números reales, es decir, NUM%(12) tendrá trece elementos de tipo entero, NUM%(0), NUM%(1), ..., NUM%(12).

En el tema siguiente se hablará de matrices multidimensionales.

Ahora puedes almacenar tus nombres y teléfonos para una guía telefónica utilizando un bucle:

```
10  REM Guia telefonica version 2
20  REM Entradas de los arrays N$ y NUM
30  CLS
40  INPUT "Numero de entradas";E
50  N=E-1
60  DIM N$(N),NUM(N)
70  FOR I=0 TO N
80  CLS
90  LOCATE ,10
100 INPUT "Nombre  ";N$(I)
110 INPUT "Numero  ";NUM(I)
120 NEXT I
```

La variable N se utiliza en la instrucción DIM de la línea 40 para asegurarnos de que hay el mismo número de elementos en las matrices N$ y NUM que los que se meten en la guía. N es un número menor que E, ya que el primer elemento de cada matriz es 0 y no 1.

Todos los nombres y números de teléfono se han metido en dos matrices. El siguiente programa visualiza la guía, y queda así:

```
10  REM Guia Telefonica Version 2
20  REM Entrada en los Arrays N$ y NUM
30  CLS
40  INPUT "Numero de entradas";E
50  N=E-1
60  DIM N$(N), NUM(N)
70  FOR I=0 TO N
80  CLS
90  LOCATE ,10
100 INPUT "Nombre    "; N$(I)
110 INPUT "Numero    "; NUM(I)
120 NEXT I
```

```
130 REM Visualiza guia
140 CLS
150 PRINT TAB(2), "NOMBRE" TAB(18) "NUMERO"
160 PRINT TAB(2) "**************************"
170 FOR I=0 TO N
180 LOCATE 2,4+I*2
190 PRINT N$(I)
200 LOCATE 18,4+I*2
210 PRINT NUM(I)
220 NEXT I
```

Cuando finalice el programa habrás perdido todos los nombres y números. Para no perderlos puedes almacenar los datos en la cinta junto con el programa utilizando los comandos READ y DATA.

Una instrucción DATA puede ir seguida de una larga lista de datos numéricos o cadenas de caracteres; los datos deben ir separados por comas:

DATA Toñi,4402073,Enrique,2246607,Tomás,4421708,

La única restricción en el número de datos almacenados es que el número total de caracteres en una línea no puede exceder de 255.

Puedes tener tantas sentencias DATA como necesites en tu programa. Sitúalas donde quieras, ya que el ordendor no accederá a las instrucciones DATA hasta que se lo indique la instrucción READ. Pero es más conveniente tenerlas todas al final del programa, ya que así es más fácil cambiar algún dato sin tener que tocar el resto del programa.

Volvamos al programa de la guía telefónica. Esta vez, en vez de introducir los datos mientras se ejecuta el programa los tendrá en instrucciones DATA. Conocemos el número de entradas de la guía, ya que acabamos de situarlas en las sentencias DATA, y no necesitamos preguntar el número de ellas. Este se asigna a la variable E, para cuando deseemos cambiar las entradas en una futura ampliación de la guía, en cuyo caso sólo se alterará la línea 30. El programa nos queda:

```
10  REM Guia Telefonica Version 2
20  REM Leer DATA y asignar Array
30  E=4
40  N=E-1
50  DIM N$(N), NUM(N)
60  CLS
70  REM Visualiza la DATA
80  PRINT TAB(2) "NOMBRE" TAB(18) "NUMERO"
90  FOR I=0 TO N
100 READ N$(I),NUM(I)
110 LOCATE 2,4+I*2
120 PRINT N$(I)
130 LOCATE 18,4+I*2
140 PRINT NUM(I)
170 NEXT I
175 END
180 DATA JUAN ,4784560, CHARO , 2241312
190 DATA "Carlota" ,7859320, CARLOS ,40 235
```


Cuando nos encontramos con la primera instrucción READ, el ordenador lee el primer elemento de la primera sentencia DATA y lo coloca en la variable de la instrucción READ. En este caso se lee JUAN y se coloca en el elemento N$(0) de la matriz N$.

La segunda vez que se encuentra la instrucción READ, el ordenador lee el segundo elemento de DATA. Una vez que haya leído todos los elementos de una instrucción DATA el ordenador continuará con la siguiente de forma secuencial.

Pensarás que es innecesario leer los datos para almacenarlos en una matriz en el mismo programa. Posiblemente estarás en lo cierto. Sin embargo, una vez que los datos se encuentran en una matriz podemos procesarlos antes de visualizarlos. En el tema siguiente se proporciona la última versión de este programa, en el que las entradas se sitúan en orden alfabético antes de visualizarlas.

Puedes utilizar más de una instrucción READ para leer una DATA. Por ejemplo:

```
10 READ A,B
20 READ A$,B$
30 PRINT A;B;A$;B$
40 DATA 10,20,"Esto es una prueba.", Hola
```

y obtendrás como resultado:

```
10   20 Esto es una prueba.Hola
```

Ten cuidado al utilizar las instrucciones DATA. Debes estar seguro de que el orden de los datos en tu sentencia corresponde con el orden de las variables descritas en la instrucción READ. Si estás leyendo una variable numérica debes tener un número en la posición correspondiente en la instrucción DATA.

En las instrucciones DATA no puedes escribir variables: sólo cadenas de caracteres y números. Las cadenas de caracteres en las instrucciones DATA no deben ir entrecomilladas, a menos que contengan algún signo de puntuación o espacios en blanco.

Si deseas leer de una instrucción DATA dos veces, la primera para que te escriba los datos y la segunda para que te realice ciertos cálculos, utiliza la instrucción RESTORE.

RESTORE motiva que el ordenador vuelva a leer los datos de la primera instrucción DATA. RESTORE, seguido por un número de línea, provoca que el ordenador vuelva a leer la primera instrucción DATA que hubiese en o después de ese número de línea. Asegúrate de que el número de línea que sigue a RESTORE es válido, es decir, que exista en el programa.

Ejecuta el siguiente programa:

```
10 READ A,B
20 PRINT "A=";A,"B=";B
30 RESTORE
```

```
40 READ C,D
50 PRINT "C=";C,"D=";D
60 DATA 10,20
```

Obtendrás:

A = 10 B = 20
C = 10 D = 20

13

Estructura y tratamiento de datos

En el tema anterior se dio un programa que almacenaba telefónos para una guía. Sería mejor escribir la guía en orden alfabético. Ya habíamos considerado una forma de ordenar listas en el tema 6. El siguiente método es, sin embargo, más flexible (no significa que sea el mejor método, sino uno de los más sencillos). A este método se le llama "método de la burbuja".

Consideremos una lista de números que deseamos ordenar:

7, 4, 5, 10, 2, 8

Los queremos ordenar en orden decreciente.

Comparamos los dos primeros números. Si el de la izquierda es menor que el de la derecha, entonces se intercambian estos dos números y se pasa a comparar el segundo con el tercero. Si los dos primeros números ya estaban en el orden correcto, no los tocamos y pasamos a comparar el segundo con el tercero.

En nuestro ejemplo, primero comparamos 7 con 4. Estos se encuentran en el orden correcto; por tanto, continuamos comparando el 4 con el 5. 4 es menor que 5, o sea, los intercambiamos. De momento, habríamos obtenido:

7, 5, 4, 10, 2, 8

Ahora se vuelve a comparar el tercer número con el cuarto; es decir, 4 con 10.

Estos también se intercambian:

7, 5, 10, 4, 2, 8

Al comparar el cuarto con el quinto, la lista no sufre modificaciones. Finalmente, comparando el quinto (número 2) con el último (número 8), los intercambianos, y nos queda:

7, 5, 10, 4, 8, 2

Aún no hemos acabado, pero observa que el número más pequeño ya está al final de la lista. Vamos a ver lo que ocurriría en la segunda pasada:

Compara	¿Intercambia?	Resultado
1.°/2.°	no	7, 5, 10, 4, 8, 2
2.°/3.°	sí	7, 10, 5, 4, 8, 2
3.°/4.°	no	7, 10, 5, 4, 8, 2
4.°/5.°	sí	7, 10, 5, 8, 4, 2
5.°/6.°	no	7, 10, 5, 8, 4, 2

Cuando acabe la segunda pasada tendremos al final de la lista a los dos números menores de ella.

El número máximo de pasadas que habría que hacer para que todos los números queden ordenados es de uno menos que el número total de elementos de la lista; en nuestro caso sería cinco el número máximo de pasadas. Sin embargo, la mayoría de las listas quedan ordenadas antes de llegar al número máximo, puesto que algunos elementos ya se encuentran ordenados en la lista inicial. Debes mirar si la lista ya está ordenada después de cada pasada.

Vamos a ordenar la guía telefónica en orden alfabético. Hay en el programa tres partes. La primera lee los datos y los almacena en una matriz; la segunda ordena los datos en orden alfabético, y la tercera visualiza los datos ordenados:

```
10  REM Guia telefonica ordenada
20  REM Leer datos de las dos matrices
30  E=4
40  N=E-1
50  DIM N$(I),NUM(I)
60  FOR I=0 TO N
70  READ N$(I),NUM(I)
80  NEXT I
90  REM Ordenacion
100 M=N-1
110 P=0
120 FOR C=0 TO M
130 IF N$(C)>N$(C+1) THEN SWAP N$(C),N$(C+1)
    : SWAP NUM(C),NUM(C+1):P=P+1
140 NEXT C
```



```
150 IF P<>0 GOTO 110
160 REM Mostrar los datos ordenados
170 CLS
180 PRINT TAB(2) "Nombre" TAB(19) "Numero"
190 PRINT TAB(2) "**************************"
200 FOR I=0 TO N
210 LOCATE 2,4+I*2
220 PRINT N$(I)
230 LOCATE 18,4+I*2
240 PRINT NUM(I)
250 NEXT I
300 DATA EDUARDO,2677668,JUANA,4326843
310 DATA ANDRES,4784560,CHARO,4775375
```

La primera y última parte de este programa ya deben resultarte familiares. (Véase el tema 12.)

De la línea 90 a la 150 contienen la rutina de ordenación. Al principio del bucle, el contador P se pone a 0. En cada bucle se compara un elemento de la matriz con el siguiente, es decir, en la primera pasada se compara N$(0) con N$(1). Si los nombres de esos elementos no están en orden alfabético, se intercambian los contenidos de los elementos. Los números de teléfono también se intercambian. P se pone a 1, para indicar que ha habido un intercambio.

En la última comparación de un bucle se comparan N$(2) con N$(3). Se intercambian el contenido de esos elementos, si es necesario, y el contador P se incrementa.

La ejecución continúa con la línea 150. Aquí el ordenador comprueba el valor de P. Si se han intercambiado algunos elementos, P será mayor que 0. El ordenador asume que los nombres no están aún ordenados y vuelve a la línea 110. P se pone a 0 y se ejecuta de nuevo el bucle de ordenación.

Si, en la línea 150, P tiene el valor 0, el ordenador sabe que no se ha intercambiado ningún elemento e indica que los nombres están en orden alfabético. El ordenador continúa con la siguiente línea del programa y visualiza la guía telefónica.

Si ejecutas este programa obtendrás:

```
NOMBRE        NUMERO
**************************
ANDRES        4784560
CHARO         4775375
EDUARDO       2677668
JUANA         4326843
```

Si quieres utilizar la misma matriz en el mismo programa, pero con distintos datos, debes borrar la matriz antigua:

ERASE N$

borra todos los nombres almacenados en la matriz N$. Con lo que puedes utilizar de nuevo la matriz N$ y redimensionarla utilizando la sentencia DIM.

Si por casualidad tienes una variable N$, no se verá afectada al utilizar la instrucción ERASE. Pero intenta no tener matrices y variables con el mismo nombre, ya que pueden confundirte.

Puedes tener matrices multidimensionales, de hasta 255 dimensiones, pero por el momento considera solamente las matrices bidimensionales. Son muy útiles para almacenar tablas:

FECHA	INGRESOS	GASTOS	SALDO
30.6.84	34.05		34.05
5.7.84		10.00	24.05
10.7.84	120.000		144.05
12.7.84		50.00	94.05
20.7.84	200.00		294.05

Estas cuatro columnas pueden almacenarse en una matriz de dos dimensiones: BANCO(4,3). Esta matriz tiene un tamaño suficiente para almacenar una tabla que consta de cinco filas y cuatro columnas, y se representa como sigue:

BANCO(0,0)	BANCO(0,1)	BANCO(0,2)	BANCO(0,3)
BANCO(1,0)	BANCO(1,1)	BANCO(1,2)	BANCO(1,3)
BANCO(2,0)	BANCO(2,1)	BANCO(2,2)	BANCO(2,3)
BANCO(3,0)	BANCO(3,1)	BANCO(3,2)	BANCO(3,3)
BANCO(4,0)	BANCO(4,1)	BANCO(4,2)	BANCO(4,3)

Al tratarse de una matriz numérica, la fecha tiene que almacenarse como un número; por ejemplo, 300684. Cada vez que se introduce algo en la tabla se almacena en un elemento separado de la matriz. La fecha 300684 se almacena en BANCO(0,0) y el último saldo, en BANCO(4,3).

El siguiente programa te muestra cómo introducir datos en una matriz de dos dimensiones. Se accede a las cuatro columnas de la matriz tomando C los valores del 0 al 3. El número de filas depende del número de transacciones que se introduzcan. Si hay T transacciones, entonces R toma los valores de 0 a T – 1. Introduce la fecha de cada transacción y las cantidades de ingresos y gastos; después de cada transacción se calcula el saldo y se almacena en la cuarta columna de la matriz. Los ingresos, gastos y saldos totales se almacenan en la última fila de la matriz.

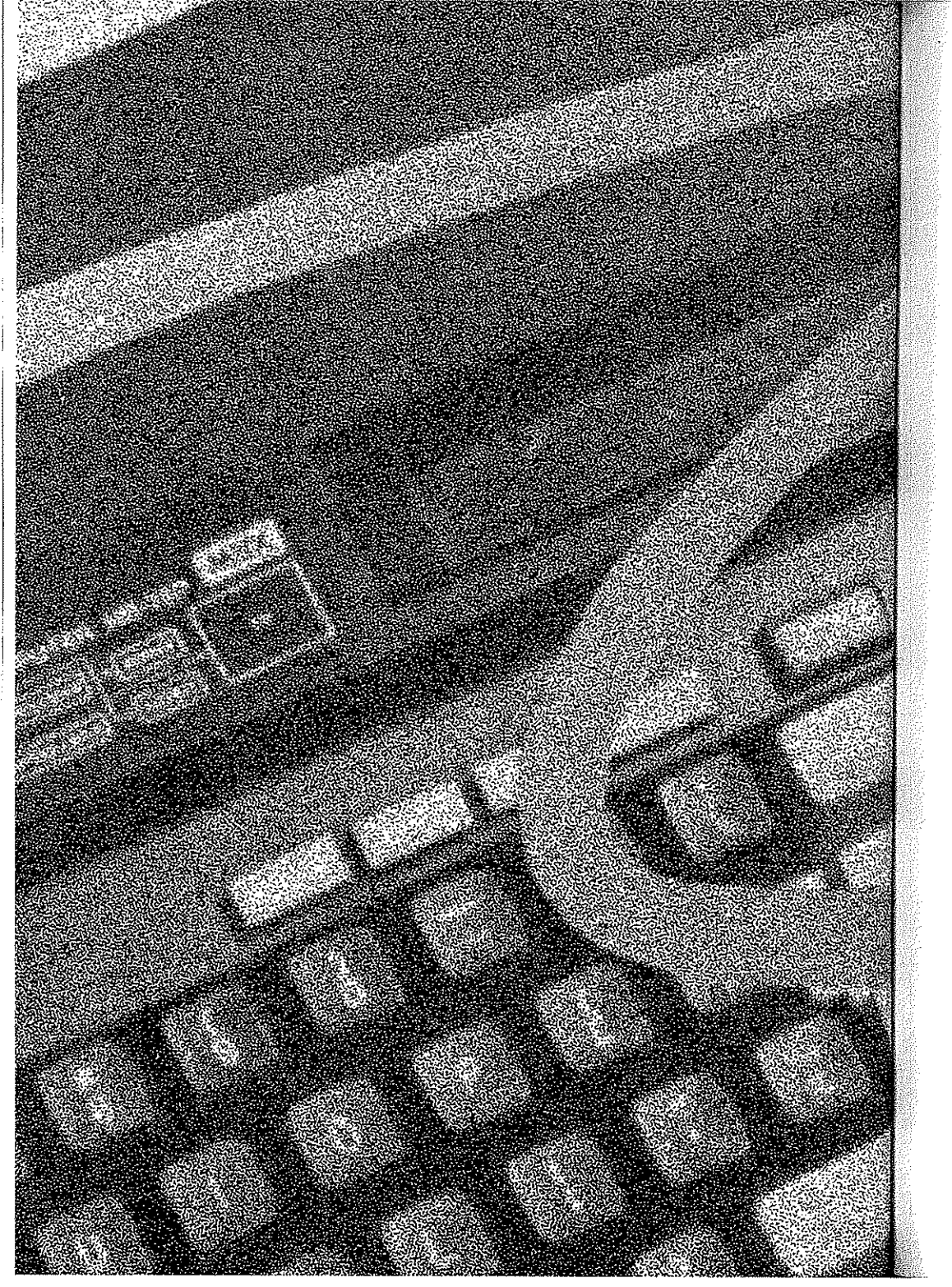
La tabla de cabeceras se almacena en una matriz de tipo cadena de caracteres. Esta matriz no precisa dimensionarse antes de su uso, ya que es de una única dimensión y tiene menos de once elementos. Todas las matrices multidimensionales deben dimensionarse antes de su utilización.

```
10  REM Balance bancario
15  REM Leer cabeceras de la matriz
20  CLS
30  FOR C=0 TO 3
40  READ CABECERA$(C)
50  NEXT C
60  REM Entrada de las transacciones
65  REM en una matriz bidimensional
70  INPUT "Numero de transacciones";T
80  DIM BANCO(T,3)
90  N=T-1
100 CLS
110 FOR C=0 TO 2
120 PRINT TAB(10*C) CABECERA$(C);
130 NEXT C
140 FOR R=0 TO N
150 FOR C=0 TO 2
160 LOCATE 10*C,R+2
170 INPUT BANCO(R,C)
180 NEXT C
190 NEXT R
200 REM Calculo del balance despues de
 las transacciones
210 BANCO(0,3)=BANCO(0,1)-BANCO(0,2)
220 FOR R=1 TO N
230 X=BANCO(R,1)-BANCO(R,2)
240 BANCO(R,3)=BANCO(R-1,3)+X
250 NEXT R
260 REM Calculo del balance total
270 FOR R=0 TO N
280 BANCO(T,1)=BANCO(R,1)+BANCO(T,1)
290 BANCO(T,2)=BANCO(R,2)+BANCO(T,2)
300 NEXT R
310 BANCO(T,3)=BANCO(T,1)-BANCO(T,2)
320 BANCO(T,0)=BANCO(N,0)
330 REM Mostrar el balance bancario
340 SCREEN 0
350 WIDTH 40
360 FOR C=0 TO 3
370 PRINT TAB(10*C) CABECERA$(C);
380 NEXT C
390 FOR R=0 TO N
400 FOR C=0 TO 3
410 LOCATE 10*C,R+2
420 PRINT BANCO(R,C)
430 NEXT C
440 NEXT R
450 PRINT "----------------------------------
    ----"
460 FOR C=0 TO 3
470 PRINT TAB(10*C) BANCO(T,C);
480 NEXT C
490 END
500 DATA FECHA,HABER,DEBE,BALANCE
```

Si introduces ingresos o gastos mayores que 99999,99, entonces desordenarás toda la tabla, ya que sólo se ha fijado un espacio de diez caracteres por entrada.

Ejercicio

1. Cambia el programa para poder introducir la fecha de la siguiente forma: 23.6.84. Necesitarás otra matriz de tipo cadena de caracteres, que puede ser FECHA$(T).



14

Las cadenas de caracteres

Hay muchas maneras de utilizar las cadenas de caracteres. Por ejemplo, puedes sumar una cadena a otra:

PRINT "Buen " + "día"

que escribirá:

Buen día

Utilizando INSTR puedes buscar una determinada cadena de caracteres dentro de otra. En un programa de juegos de aventuras quieres preguntar al jugador dónde quiere ir. El juego tomará diferentes caminos, dependiendo de si se escribe Norte, Sur, Este u Oeste.

Este fragmento de programa te indica cómo hacerlo:

```
100 INPUT "Donde vas ahora";B$
110 IF B$="Norte" GOTO 200
120 IF B$="Sur" GOTO 300
130 IF B$="Este" GOTO 400
140 IF B$="Oeste" GOTO 500
150 PRINT "INTENTALO DE NUEVO": GOTO 100
```

Al contestar el jugador puede haber escrito toda una frase; por ejemplo: "Voy al Este".

Tal como está hecho el programa, no se ejecutará ninguna rama y se le preguntará al jugador de nuevo hasta que introduzca la palabra correcta como "Este" y continuará con el juego. Sería mejor que el ordenador examinara la cadena de caracteres que se ha introducido y ver si reconoce algún fragmento de ella. Esto es lo que hace la instrucción INSTR. El siguiente ejemplo te muestra cómo se utiliza:

```
100 INPUT "Donde vas ahora";B$
110 IF INSTR(B$,"NORTE")<>0 GOTO 200
120 IF INSTR(B$,"SUR")<>0 GOTO 300
130 IF INSTR(B$,"ESTE")<>0 GOTO 400
140 IF INSTR(B$,"OESTE")<>0 GOTO 500
150 PRINT "INTENTALO DE NUEVO": GOTO 100
```

En la línea 110, el comando INSTR busca la cadena de caracteres "NORTE" dentro de la cadena principal, y si la encuentra, devuelve el número de la posición donde se encuentra el primer carácter de "NORTE" en la cadena principal. En caso contrario, INSTR devuelve un cero; si B$ fuese:

"ME VOY AL NORTE AHORA"
1 2345678901

entonces INSTR(B$,"NORTE") devuelve el número 11, ya que la N de NORTE ocupa la posición del carácter decimoprimero. Para comprobarlo escribe el siguiente programa:

```
10  A=INSTR("ME VOY AL NORTE","NORTE")
20  PRINT A
```

Se escribe el número 11. INSTR siempre devuelve un número, por lo que se debe asignar a una variable numérica.

Ten presente el incluir la cadena de caracteres de la instrucción INSTR entre comillas.

Cuando se busca una cadena de caracteres puedes empezar desde un punto especificado:

INSTR(10,A$,B$)

empieza a buscar B$ en A$ desde la posición de carácter número 10 en adelante.

Este comando es útil en el siguiente programa, que busca en la palabra CONSTANTINOPLA la letra N:


```
10 A$="CONSTANTINOPLA"
20 B$="N"
30 C=0
40 C=INSTR(C+1,A$,B$)
50 IF C=0 THEN END
60 PRINT "hay una ";B$" en la posicion ";C
70 GOTO 40
```

En este programa la expresión C + 1 se utiliza para inicializar dónde va a comenzar la búsqueda. La primera vez que el comando INSTR encuentre entre una "N" es cuando C = 3, y la siguiente búsqueda empieza en la posición 4. Después de que se haya encontrado la tercera "N", para C = 10 la cuarta y última búsqueda empieza en la posición 11. No hay más "N"; por tanto, C se vuelve a poner a 0, originando la finalización del programa en la línea 50.

Siempre que intentes buscar una cadena de caracteres larga en una más pequeña la instrucción INSTR devolverá un cero:

INSTR("día", "buen día")

devolverá un cero.

Ahora veremos tres funciones similares:

RIGHT$, LEFT$ y MID$

Primero veremos RIGHT$. Aquí tienes un ejemplo:

```
10 A$=RIGHT$("Que tiempo tan horrible hace",4)
20 PRINT A$
```

da como resultado:

hace

Se devuelven los cuatro caracteres de la derecha en la variable A$.

Aquí tienes un ejemplo del funcionamiento de LEFT$:

```
10 A$=LEFT$("Que tiempo tan horrible hace"
,10)
```

da como resultado:

Qué tiempo

Finalmente MID$:

```
10 A$=MID$("Que tiempo tan horrible hace",12,17)
20 PRINT A$
```

que da como resultado:

tan horrible hace

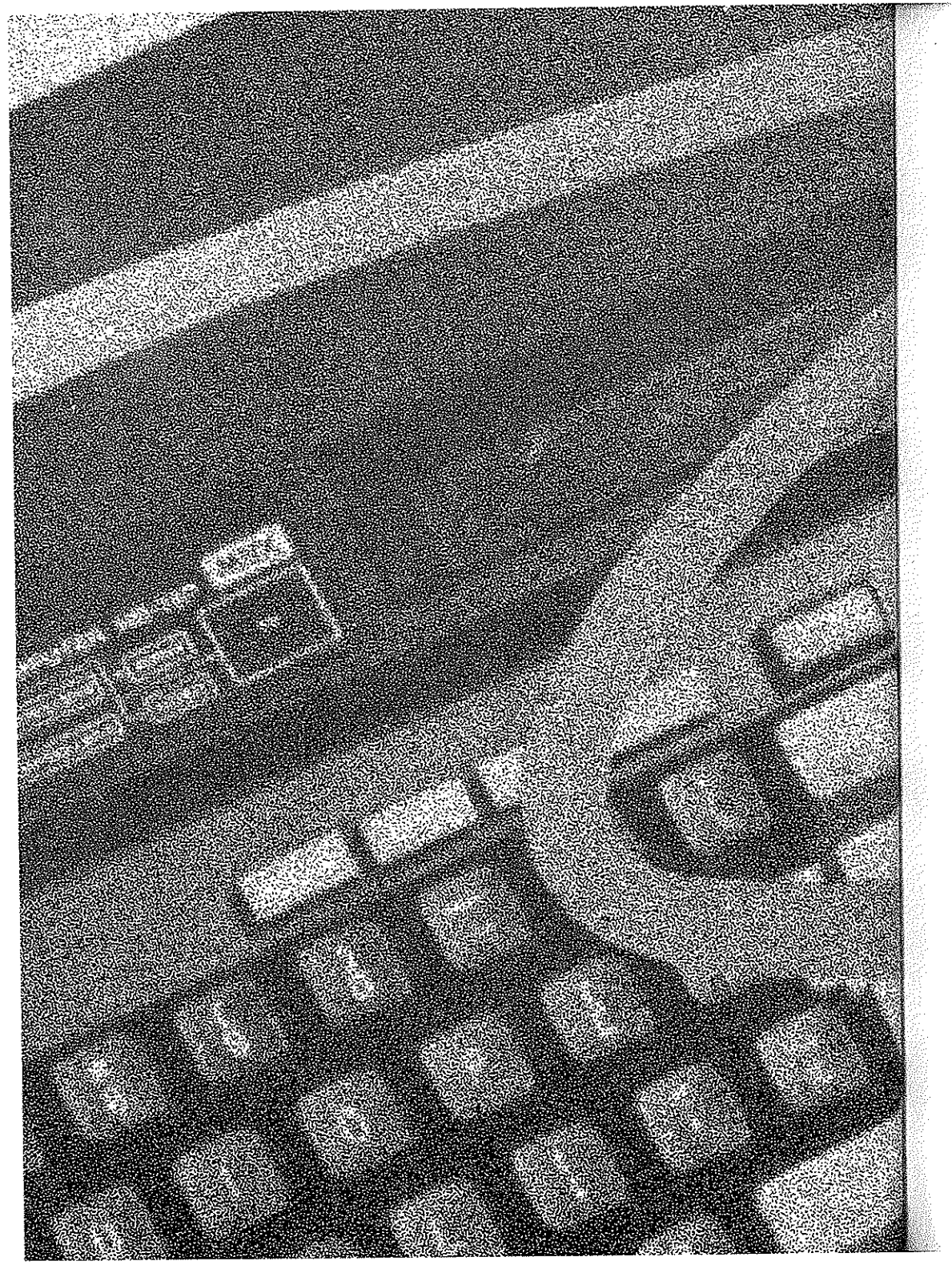
El primer número de la instrucción MID$ proporciona la posición del primer carácter que se tiene que devolver. El segundo indica cuántos caracteres debe devolver. Por lo que en el ejemplo anterior almacena en la variable A$ 17 caracteres de los 28 dados, a partir de la posición 12. Recuerda que no puedes tener una línea con una longitud mayor de 255 caracteres. El argumento utilizado en estas funciones debe estar comprendido entre 1 y 255.

Si el primer argumento de MID$ fuese mayor que el número de caracteres de la cadena obtendrás una cadena de caracteres vacía.

Otra vez no olvides las comillas en una cadena de caracteres y no intentes igualar una de estas tres instrucciones a una variable numérica; debes utilizar las variables de tipo cadena de caracteres con estas funciones.

Ejercicios

1. Utilizando INSTR, busca el resultado de:
   a) Buscar una cadena de caracteres vacía, o sea, "".
   b) Buscar una cadena de caracteres dentro de otra vacía.
   c) Buscar una cadena de caracteres vacía dentro de otra vacía.
2. Escribe por separado las palabras "¿Hay alguien ahí?". Puedes utilizar INSTR para encontrar la posición de cada palabra y luego utiliza RIGHT$, LEFT$ y MID$.
3. Cambia el último programa para que puedas separar las palabras de cualquier cadena de caracteres que introduzcas.



15
Funciones

Ya te has encontrado con unas cuantas funciones; por ejemplo:

```
INT       TEMA 4
INSTR     TEMA 14
LEFT$     TEMA 14
MID$      TEMA 14
```

Hablando de forma general, cada una de ellas actúa del mismo modo. Tienes un objeto llamado argumento, sobre el cual la función actúa y produce un resultado. Una vez que se ha definido una función, ésta produce los mismos resultados, siempre que se le asigne el argumento apropiado:

```
INT(x)
```

Aquí x es el argumento. El argumento de una función siempre se encierra entre paréntesis. El argumento debe ser un número, una variable numérica o una expresión. Siempre que x se encuentre dentro de este rango INT devolverá el valor entero de x después de truncarlo al número entero menor más cercano.

```
INT(7.4)      devuelve 7.
INT(-2.3)     devuelve -3.
```

Si te confundes con los números negativos mira la siguiente tabla:

... −6 −5 −4 −3 −2 −1 0 1 2 3 4 5 6...

Los números están en orden creciente, de izquierda a derecha. Por eso puedes ver que −3 es el entero más cercano a −2.3.

Hay una función muy similar a INT, llamada FIX.

FIX también tiene argumento real y devuelve un entero. En el caso de los números positivos devuelve el menor número entero más cercano al igual que INT, pero para números negativos devuelve el mayor número entero más cercano, a diferencia de INT:

FIX(5.5) devuelve 5.
FIX(−2.2) devuelve −2.

Hay dos funciones más que también tienen argumentos numéricos ABS y SGN.

SGN devuelve el signo del argumento:

- Si el argumento es un número negativo, la función devuelve −1.
- Si el argumento es cero, la función devuelve 0.
- Si el argumento es un número positivo, la función devuelve 1.

```
10 A=SGN(-402.2)
20 PRINT "SGN(-402.2)=";A
```

que da:

SGN(−402.2)= −1

Por otro lado, ABS se utiliza para convertir los números negativos en positivos. A los números positivos los deja tal como estaban:

```
10 A=ABS(-402.2)
20 PRINT "ABS(-402.2)=";A
```

que da:

ABS(−402.2)=402.2

Los argumentos de estas dos funciones pueden ser tanto expresiones como variables.

SGN(4 + 3 * −2) devuelve −1.
ABS(porciento) convierte el número de la variable numérica porciento en un número positivo.

Aquí tienes un programa corto que te muestra una posible utilización de ABS:

```
10 REM Calculo de tu edad
20 INPUT "Fecha de hoy: DIA, MES, AÑO";DH,MH,
   AH
30 INPUT "Fecha de nacimiento: DIA, MES, AÑO"
   ;DN,MN,AN
40 DA=ABS(DH-DN)
50 MA=ABS(MH-MN)
60 AA=ABS(AH-AN)
70 PRINT "Tienes";AA;" AÑOS,";MA;"Meses,";DA;
" Dias."
```

Ahora tenemos dos funciones que tratan ambas con cadenas de caracteres y números: VAL y STR$.

Si una cadena de caracteres contiene un número, entonces VAL asignará dicho número a una variable numérica, es decir, suprime las comillas y omite los blancos de la cadena. Por ejemplo:

```
A = VAL("  273")
PRINT "A=";A
```

obteniendo:

A=273

Si hay más de un número en la cadena de caracteres, VAL sólo devolverá el primero que encuentre. Por esta razón VAL no evalúa expresiones dentro de cadenas de caracteres:

PRINT VAL("3+4=7")

que dará:

3

No olvides que aunque el argumento de VAL es una cadena de caracteres el resultado que devuelve VAL es un número y se debe asignar a variables numéricas:

A$=VAL(7) ES INCORRECTO

VAL no puede utilizarse con cadenas que contengan números entre letras:

```
10 A=VAL("Hoy es mi duodecimo cumpleaños")
20 PRINT A
```

obtendrás:

0

La función inversa a VAL es STR$, que convierte un argumento numérico en una cadena de caracteres:

```
A$="Tengo "+STR$(60)+" años"
PRINT A$
```

obtendrás:

tengo 60 años

No olvides que, aunque el argumento es un número, el resultado es una cadena de caracteres; por tanto, debes asignarle una variable del mismo tipo.

En el tema 13 utilizamos una matriz numérica llamada BANCO para almacenar fecha, ingresos, gastos y saldos. La fecha tenía que introducirse como un número, es decir, 240684 en lugar de 24/6/84. Para introducirlo de la segunda forma debes almacenar las fechas por separado en otra matriz de tipo cadena de caracteres, por lo que el programa utilizará una matriz numérica para las transacciones y dos matrices de tipo cadena de caracteres para la cabecera y las fechas. Utilizando STR$, todos los datos podrán almacenarse en una única matriz de tipo cadena de caracteres. Para hacerlo tenemos que realizar las siguientes modificaciones en el programa:

```
170 INPUT BANCO
180 BANCO$(R,C)=STR$(BANCO)
```

Para calcular el nuevo saldo después de cada transacción debes evaluar las cadenas de caracteres de las columnas de ingresos y gastos utilizando VAL, y convertir la respuesta en una cadena de caracteres:

```
210 BANCO$(0,3)=STR$(VAL(BANCO$(0,1))-VAL(BAN
CO$(0,2)))
```

¡Quizá desees volver al programa de BALANCE DE BANCO y alterarlo!

La última función que actúa sobre cadenas de caracteres es LEN. LEN devuelve la longitud de la cadena de caracteres que se da como argumento:

```
10 A$="CONSTANTINOPLA"
20 A=LEN(A$)
30 PRINT "La longitud de la palabra ";A$;" es
 de ";A;" caracteres"
```


Recuerda que aunque la función tiene como argumento una cadena de caracteres, siempre devuelve un número, por lo que esta función debe asignarse a una variable numérica.

Finalmente, una función algo distinta es RND.

RND genera un número aleatorio de catorce dígitos entre 0 y 1. De hecho, el número generado no es verdaderamente aleatorio, ya que sigue una secuencia fijada. Sin embargo, como la secuencia es muy extensa, se puede considerar al número como generado aleatoriamente.

Puedes tener tres tipos de argumentos con RND:

Si el argumento es positivo el número generado por RND en el programa es el siguiente en la secuencia. El valor con que se inicia la secuencia es siempre el mismo y el ordenador lo pone a cero al final del programa. Escribe:

```
10 PRINT RND(1)
20 PRINT RND(1)
30 PRINT RND(1)
```

Cada vez que ejecutes el programa obtendrás la siguiente secuencia:

```
.59521943994623
.10658628050158
.76597651772823
```

No tiene ningún efecto el cambiar el argumento por otro número positivo.

Si el argumento es cero, el número aleatorio generado es el mismo que el último. Por ejemplo:

```
10 X=RND(1)+RND(1)
20 PRINT "El segundo numero aleatorio generad
o es ";RND(0)
30 PRINT "El primer numero aleatorio generado
 es ";X-RND(0)
```

Si el argumento es negatuvo, la parte de la secuencia de la que se toma el número aleatorio se fija según el valor del argumento:

```
10 A=RND(-3)
20 B=RND(1)
30 PRINT A,B
```

obteniendo la secuencia:

```
.84389820420821
.2962486816692
```

cada vez que se ejecuta el programa. Ahora sustituye −3 por −4 y obtendrás una secuencia distinta. Cambiar. el argumento positivo no tiene ningún efecto.

Un número aleatorio entre 0 y 1 no tiene mucha utilidad. Preferirás tener un número aleatorio entre 1 y 6. Por ejemplo, este programa simula un lanzamiento de dados:

```
10 REM lanzamiento de dados
20 A=INT(6*RND(1)+1)
30 PRINT "Te ha salido un ";A
```

Sin embargo obtendrás el mismo número cada vez que tires el dado. Si deseas obtener un número aleatorio distinto cada vez, pon la línea:

RDRST=RND(−TIME)

al principio de tu programa.

La función TIME devuelve el valor del reloj interno del ordenador. Cuando conectes el ordenador, el reloj se fija a cero y se va incrementando cada 1/50 de segundo; es decir, que TIME será igual a 50 cuando haya pasado un segundo. Por lo que, utilizando TIME como argumento de la función RND, hará que la secuencia de números aleatorios se lea desde un punto diferente cada vez que se encuentra la función RND en el programa.

¿Has visto cómo utilizar esto junto con la última función en el juego? Te proponemos que hagas un programa para jugar al ahorcado.

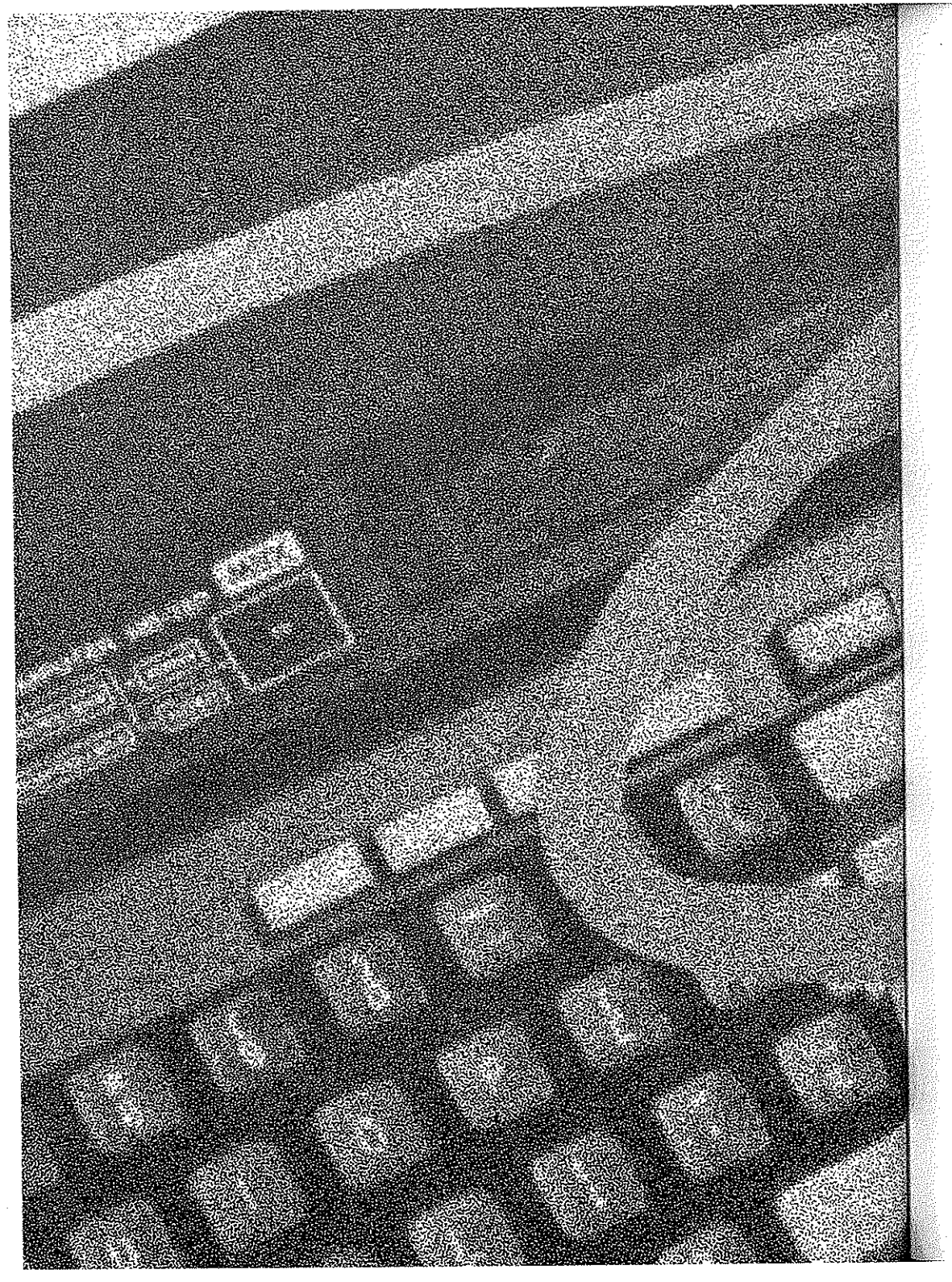
El juego debe seguir la siguiente estructura:

1. Leer aleatoriamente de una instrucción DATA (estas instrucciones pueden ponerse juntas al final de programa). Para leer (READ) de una línea aleatoria utiliza RESTORE, seguida por un número de línea elegido al azar. Piensa en cómo acceder a más de una línea de una instrucción DATA.
2. Una vez que hayas leído la palabra dile al jugador su longitud utilizando LEN.
3. Pregunta al jugador por una letra.
4. Utilizando INSTR, busca si la letra se encuentra en la palabra (consultar el tema 14 por si se te ha olvidado cómo hacerlo).
5. Cuenta el número de intentos del jugador.

6. Si el jugador no ha adivinado la palabra después de diez intentos, entonces termina el juego: ha ganado el ordenador.
7. Pregunta al jugador si desea jugar de nuevo utilizando INKEY$.

Ejercicio

1. Escribe el programa del ahorcado.



16

Tus propias funciones

Vamos a definir nuestras propias funciones utilizando DEF FN. Una vez que hayas definido una función y la hayas dado un nombre, puedes utilizarla en cualquier parte del programa.

Primero veremos un ejemplo matemático.

Supón que quieres definir una función que calcule el cuadrado de un número, es decir, elevar un número a la potencia de dos, que es lo mismo que multiplicar al número por sí mismo.

Llamaremos a la función CUADRADO y la definiremos como:

```
DEF FNCUADRADO(X)=X^2
```

Advierte que el nombre de la función va inmediatamente después del comando DEF FN.

A X se le denomina "falso parámetro". Sólo lo utilizas para mostrar qué es lo que hace la función. Cuando utilices la función debes sustituir el falso parámetro por un parámetro real. El parámetro real tiene que ser del mismo tipo que el falso parámetro. En el ejemplo anterior el parámetro real puede ser tanto una variable numérica como un número. Cuando se define una función se deben incluir a continuación del nombre de la función todos los parámetros entre paréntesis.

Vamos a utilizar esta función para calcular el cuadrado de 13 y asignar el

resultado a la variable R. Para hacerlo utilizaremos el comando FN seguido del nombre de la función y del parámetro real entre paréntesis:

R=FNCUADRADO(13)

Al ejecutar esto el ordenador buscará la definición de la función CUADRADO. Cuando la encuentra sustituye el falso parámetro por el parámetro real 13. El resultado evaluado se asigna a la variable R.

A X también se le denomina "variable local", que significa que únicamente puede actuar localmente; es decir, dentro de la función:

```
10 X=4
20 DEF FNCUADRADO(X)=X^2
30 R=FNCUADRADO(12)
40 PRINT "R= ";R;"X= ";X
```

obtendrás:

R=144 X=4

En una función se puede utilizar más de un parámetro. Todos ellos deben ir declarados en la definición e ir separados por comas. Cuando llames a la función debes pasar el mismo número de parámetros reales, del mismo tipo que los falsos parámetros de la definición.

El siguiente programa eleva un número a la potencia de otro número; ambos se pasan en la definición de la función:

```
10 DEF FNELEVA(X,Y)=X^Y
20 INPUT "Base ";A
30 INPUT "Exponente";B
40 R=FNELEVA(A,B)
50 PRINT A;"^";B;"=";R
```

La definición de la función debe caber en una línea del programa.

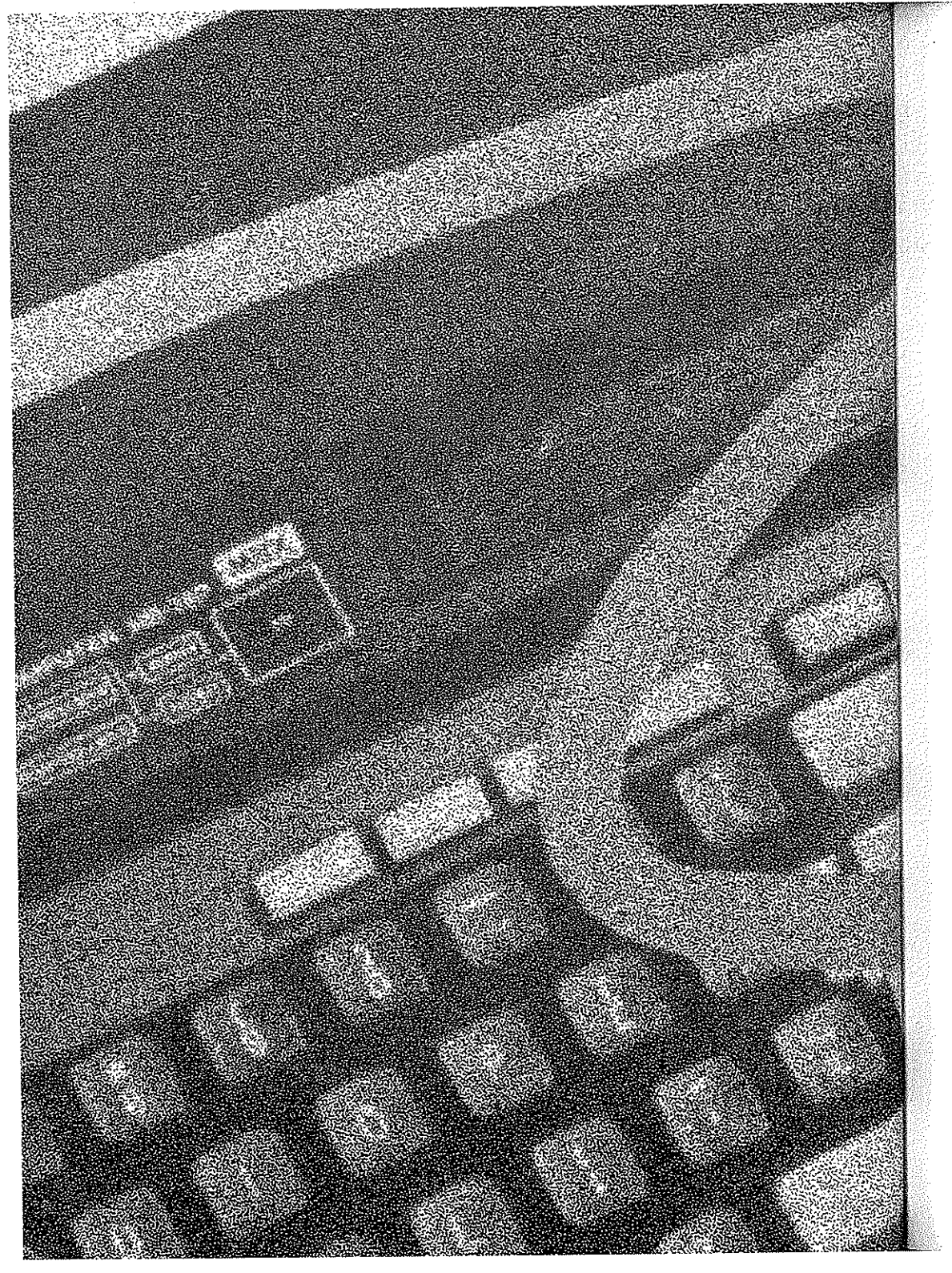
Ahora veamos otro ejemplo utilizando cadenas de caracteres. Esta función escribe el primer carácter de una cadena de caracteres:

```
10 DEF FNST$(A$)=RIGHT$(A$,LEN(A$)-1)
20 INPUT "Cadena de caracteres";S$
30 PRINT S$,FNST$(S$)
```


Como esta función devuelve un carácter, el nombre de la función debe finalizar con el signo $. Cualquier nombre de variable válido se aceptará como nombre de función.

Ejercicio

1. Define una función que suprima los cuatro últimos caracteres de una cadena.



17

Estructura de tus programas

A veces te encuentras con que necesitas efectuar la misma operación muchas veces en diferentes puntos del programa y tienes que repetir varias líneas frecuentemente; es mejor que esas líneas las sitúes en lo que se denomina "subrutina".

Cuando necesites en el programa esas líneas saltarás a la subrutina utilizando la instrucción GOSUB. Cuando se hayan ejecutado todas las líneas de la subrutina se retorna al programa principal. Una subrutina puede contener tantas líneas como se necesiten. En el siguiente ejemplo la subrutina se encuentra entre las líneas 100 y 130:

```
10 CLS
20 PRINT"¿Te gusto?"
30 GOSUB 110
40 PRINT "¿Crees que soy inteligente?"
50 GOSUB 110
60 PRINT "¿Estoy bueno?"
70 GOSUB 110
80 PRINT "¡Me estas haciendo la pelota!"
90 END
100 REM Subrutina
110 INPUT "Si o No";SN$
120 IF SN$="No" THEN PRINT "¡¡¡Me has matado!!!":END
130 RETURN
```

Observa la instrucción RETURN que se encuentra al final de la subrutina. Es muy importante, ya que indica al sistema que retorne al programa principal. La ejecución continuará en la siguiente instrucción a la llamada GOSUB.

Una buena idea es situar un comentario (REM) justo antes del comienzo de la subrutina para distinguirlo del programa principal. No utilices el número de línea de la instrucción REM como referencia para llamar a la subrutina, ya que, en caso de que el programa sature la memoria, deberás eliminar estas líneas con comentarios (REM).

Asegúrate de que el ordenador no ejecuta la subrutina por accidente, poniendo al final del programa principal, justo antes de las subrutinas, una instrucción END.

Otro uso generalizado de la instrucción GOSUB es en las sentencias IF... THEN... ELSE. Como esta instrucción tiende a ser muy larga y complicada, se simplifica con el uso de subrutinas.

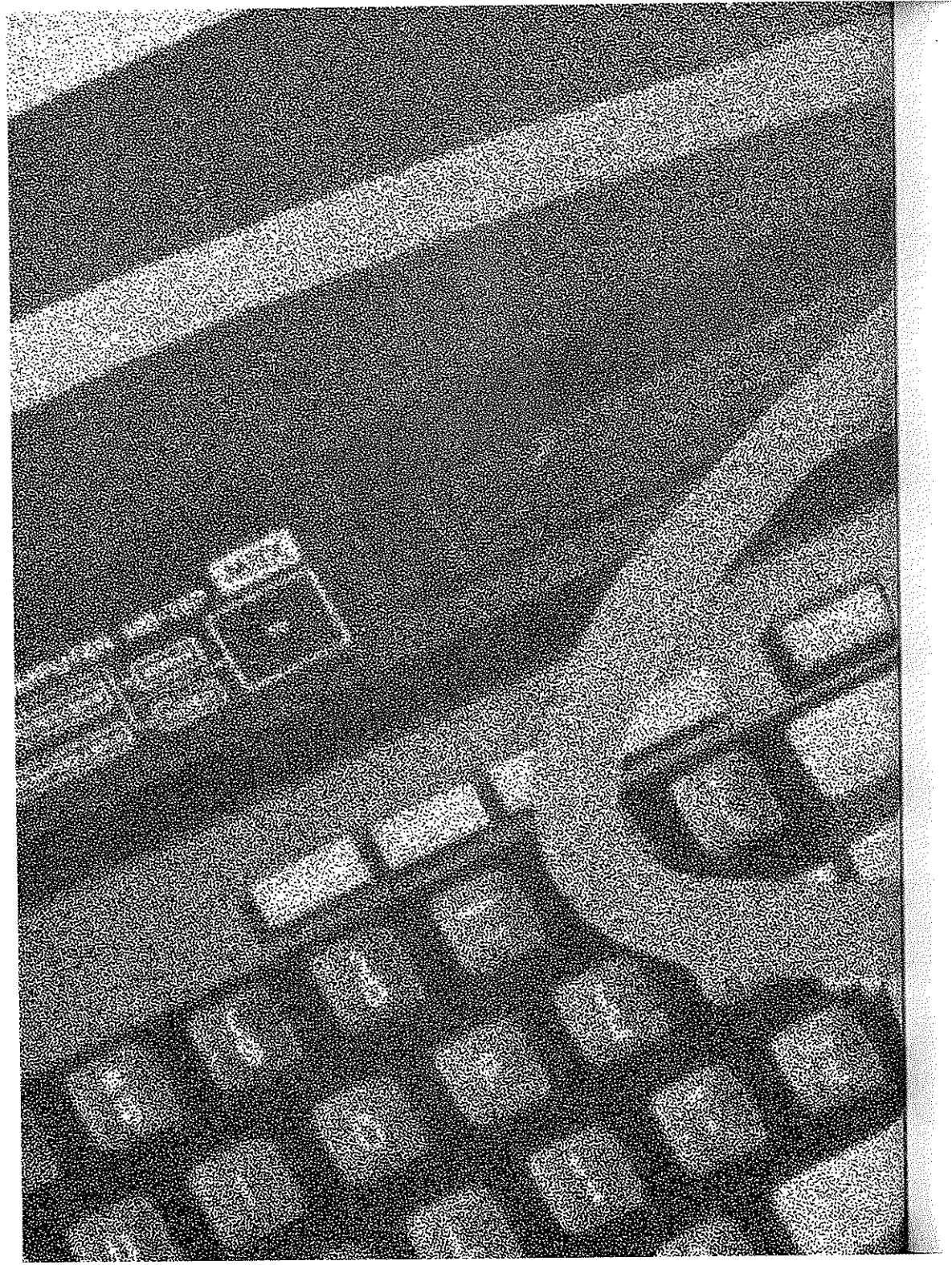
El siguiente programa pide que se le introduzcan dos números positivos distintos. La instrucción IF... THEN... ELSE se utiliza para efectuar comprobaciones sobre estos números. Si son distintos y positivos se escribe su producto y su suma; en caso contrario se pide que se intente de nuevo:

```
10 INPUT "DOS NUMEROS POSITIVOS Y DISTINTOS";
A,B
20 IF A=B OR A<0 THEN PRINT "INTENTALO DE NUE
VO" ELSE PRINT "A=";A;"B=";B: PRINT "A+B=";A+
B: PRINT "A*B=";A*B
30 GOTO 10
```

Ahora, utilizando una subrutina:

```
10 INPUT"Dame dos numeros positivos";A,B
20 IF A=B OR A<0 OR B<0 THEN PRINT "Intentalo
de nuevo" ELSE GOSUB 100
30 GOTO 10
90 REM Subrutina
100 PRINT "A=";A,"B=";B
110 PRINT "A+B=";A+B
120 PRINT "A*B=";A*B
130 RETURN
```

el programa es más largo, pero más sencillo de leer.



18

Programas de bifurcación

En algún punto de tu programa principal querrás que te ofrezcan un determinado número de opciones. Por ejemplo, las primeras líneas de tu programa menú podrían mostrarse por pantalla lo siguiente:

```
**********MENU**********
1. Visualiza la guía telefónica.
2. Entrada de datos a la guía.
3. Supresión de datos de la guía.
4. Fin.
******************************
```

La continuación de este programa sería algo parecido a:

```
100 A$=INKEY$
120 IF A$="1" THEN GOSUB 1000
130 IF A$="2" THEN GOSUB 2000
140 IF A$="3" THEN GOSUB 3000
150 IF A$="4" THEN GOSUB 4000
160 GOTO 100
```

Se incluye la línea 160 en previsión de que se haya pulsado una tecla distinta a 1, 2, 3 ó 4.

Existe una forma más corta para escribirlo, utilizando ON... GOSUB. Mediante esta instrucción puedes saltar al número de subrutina que desees. En el ejemplo anterior queríamos saltar a una de cuatro subrutinas, por lo que escribiremos los números de línea de las cuatro subrutinas a continuación de la palabra GOSUB:

ON...GOSUB 1000,2000,3000,4000

Ahora debemos programar a qué subrutina queremos saltar. Esto se hace situando un número o una expresión a continuación de ON y antes de GOSUB. Si la expresión da como resultado un 1, la ejecución continúa en la primera subrutina especificada a continuación de GOSUB; si evalúa 2, la ejecución saltará a la segunda subrutina especificada, etc.

Luego podemos sustituir las líneas 100 a 160 por:

```
100 A$=INKEY$
110 A=VAL(A$)
120 ON A GOSUB 1000,2000,3000,4000
130 GOTO 100
```

Observa que la cadena de caracteres A$ se ha cambiado por una variable numérica A, antes de utilizarla en la instrucción ON...GOSUB; en caso contrario se hubiese producido un error, ya que después de la palabra ON no pueden ir más que variables numéricas.

Cuando la subrutina finaliza el ordenador volverá a la línea 130.

Si la variable numérica A hubiese contenido un número real, por ejemplo 2.9, el ordenador lo hubiera truncado al menor entero y hubiera saltado a la segunda subrutina.

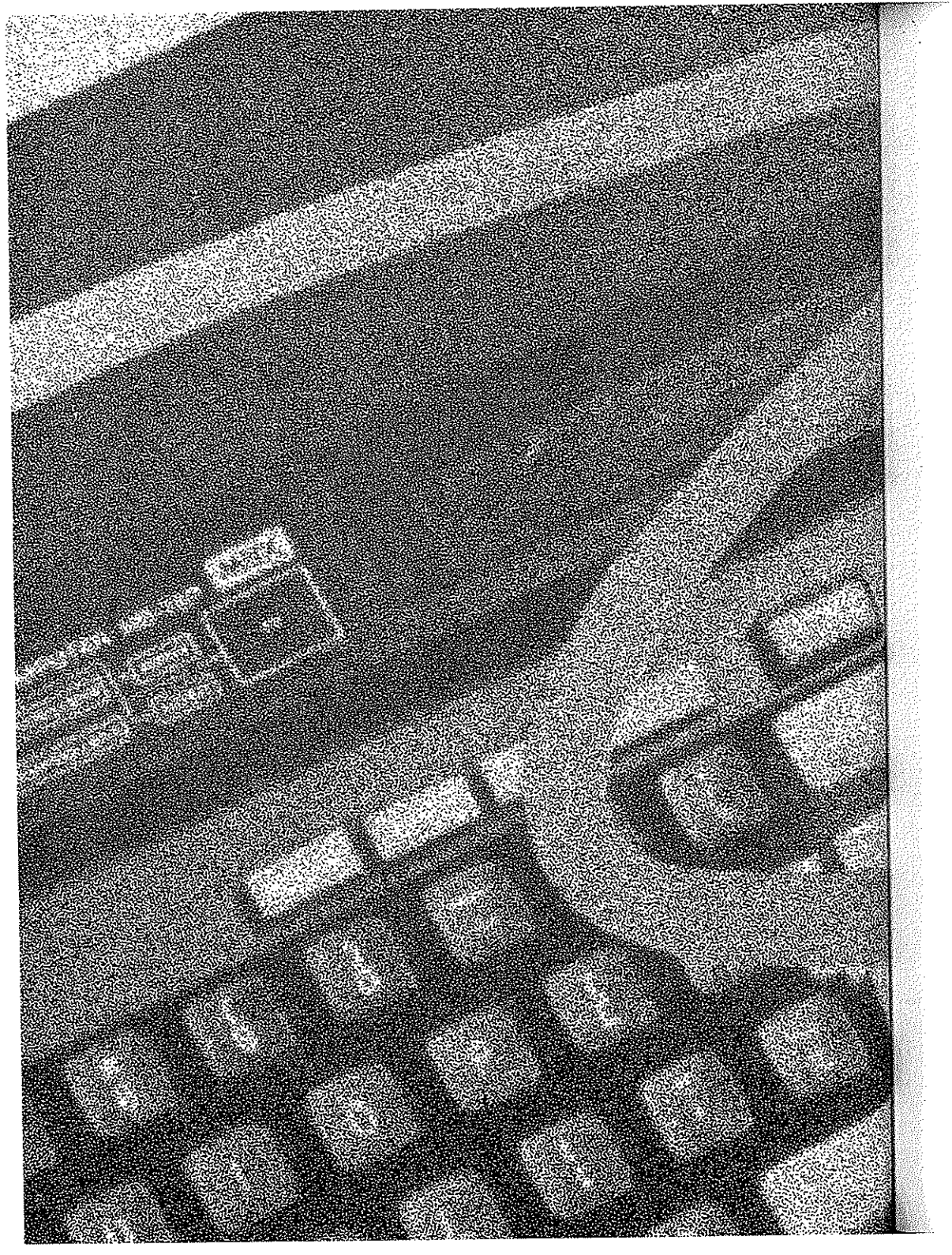
ON...GOTO tiene el mismo efecto que ON...GOSUB. En este caso el ordenador salta a un número de línea de la lista que corresponde al número que sigue a la instrucción ON. La ejecución del programa continúa desde este número de línea:

```
10 INPUT "1,2 o 3";A
20 ON A GOTO 370,420,10
```

Si

A=1 la siguiente línea a ejecutar es la 370.
A=2 la siguiente línea a ejecutar es la 420.
A=3 la siguiente línea a ejecutar es la 10.

Nota: Los números de línea de la lista no tienen por qué estar en orden numérico.



19

Funciones matemáticas

Si tu fuerte no son las matemáticas y encuentras este tema muy pesado, pásalo y continúa con el siguiente.

Este tema trata de funciones matemáticas tales como:

^, SQR
TAN, SIN, COS y ATN
EXP y LOG

Ya sabes cómo elevar un número a una potencia utilizando el signo:

4^2 = 16 (normalmente se escribe $4^2 = 16$)

En general, elevar un número "n" a una potencia "p" equivale a multiplicar "n" por sí mismo "p" veces. En el ejemplo anterior, n = 4 y p = 2. Por lo que 4^2 es equivalente a 4 * 4.

La función raíz cuadrada, **SQR**, te proporciona la raíz cuadrada de un número. Es el inverso de elevar un número al cuadrado.

Si elevas cinco al cuadrado (5^2) obtienes 25; si ahora efectúas la raíz cuadrada de 25, vuelves a obtener 5:

```
10 A=SQR(25)
20 PRINT "La raiz cuadrada de 25 es ";A
```

obtendrás:

La raíz cuadrada de 25 es 5

El argumento de la función SQR debe ser un número positivo o una expresión que al evaluarse dé un número positivo (el ordenador no trabaja con números complejos).

¿Qué ocurrirá si elevas un número a una potencia negativa? Es fácil averiguarlo. Escribiendo:

```
PRINT 4^-1
```

obtienes:

```
.25
```

que es 1/4; en general:

$n^{-p} = 1/n^{p}$

Vamos a ver qué ocurre cuando elevamos un número a una potencia decimal o una fracción. Escribe:

```
PRINT 4^0.5
```

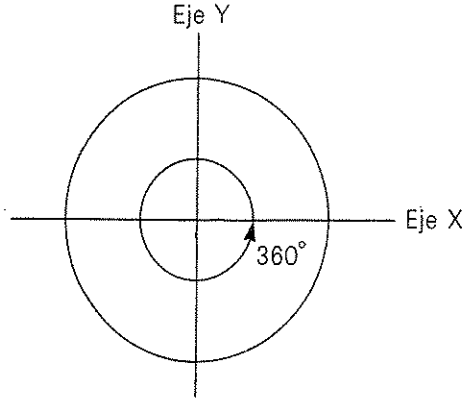
te da como resultado 2.

$4^{0.5}$ es equivalente a $4^{1/2}$, así en general:
$n^{1/p}$ nos dará la raíz p-ésima de n.

En el ejemplo, n = 4, p = 2, obtienes la raíz cuadrada de 4, que es 2. Elevar un número a la potencia 1/2 es equivalente a hacer la raíz cuadrada del número.

SIN, COS, TAN y **ATN** son funciones trigonométricas.

Los argumentos de estas funciones deben ser ángulos. Posiblemente pienses que los ángulos vengan expresados en grados:



Un círculo tiene 360°. Los ángulos se miden por medio del eje X y del eje Y. El eje X divide el círculo en dos partes iguales y va desde la posición de las nueve en punto a la posición de las tres en punto. El eje Y divide al círculo en dos partes iguales, desde la posición de las doce en punto a la posición de las seis en punto. El ángulo entre estos dos ejes es de 90°.

Sin embargo, el ordenador mide los ángulos de una manera distinta. Para entenderlo, considera el círculo con un radio de una unidad. La unidad no importa aquí, pero puede ser 1 mm, 1 m o 1 km.

La siguiente fórmula te proporciona la longitud del círculo:

$C = 2 * PI * r$

C = Circunferencia.
PI es un número especial utilizado en matemáticas, y es igual a 3,1415926...
r es el radio del círculo.

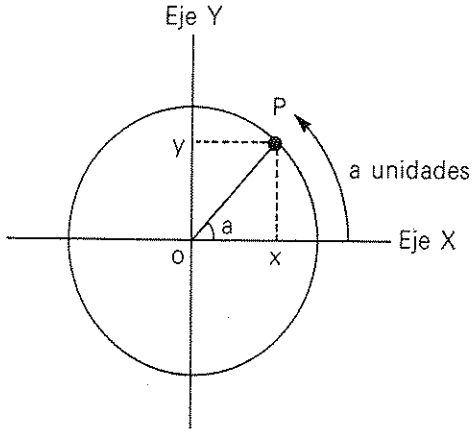
Por tanto, para nuestro círculo, y como r = 1:

$C = 2 * PI$

Mirando esta otra definición podemos decir que la circunferencia de nuestro círculo tiene un ángulo de $2 * PI$ radianes en el centro del círculo. Por lo que:

$2 * PI$ radianes es equivalente a 360°.
En radianes, el ángulo entre los ejes X e Y es de PI/2.

Si ahora tenemos un punto en algún lugar del círculo, podemos encontrar su valor utilizando las funciones SIN y COS.



El punto puede estar en la posición P, y se ha movido un ángulo de "a" radianes a partir del eje X o, si lo prefieres, se ha movido una distancia de "a" unidades alrededor de la circunferencia.

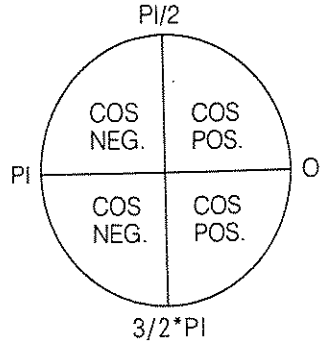
Si a partir del punto P trazas unas líneas perpendiculares que corten a los ejes X e Y, obtendrás las posiciones de X e Y utilizando COS y SIN, respectivamente.

```
X = COS(a)      COS es el coseno del ángulo a.
Y = SIN(a)      SIN es el seno del ángulo a.
```

Cuando medimos a lo largo del eje X consideramos X positivo si estamos a la derecha del eje Y, y X negativo si estamos a la izquierda del mismo eje. Por tanto, se puede asegurar que:

El COS de cualquier ángulo entre 0 y PI/2 es positivo.
El COS de cualquier ángulo entre PI/2 y 3 * PI/2 es negativo.
El COS de cualquier ángulo entre 3 * PI/2 y 2 * PI es positivo.

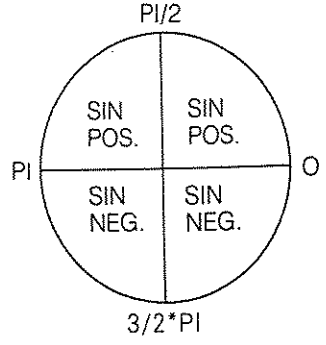
Es decir, para el COS:



Cuando medimos a lo largo del eje Y, Y es positivo por encima del eje X y negativo por debajo. Se puede asegurar, por tanto, que:

El SIN de cualquier ángulo entre 0 y PI es positivo.
El SIN de cualquier ángulo entre PI y 2 * PI es negativo.

Es decir, para el SIN:



Si damos más de una vuelta al círculo, COS y SIN seguirán conservando los mismos valores que en la primera vuelta.

```
SIN(a + 2 * PI) = SIN(a)
COS(a + 2 * PI) = COS(a)
```

La tangente de un ángulo es:

```
TAN(a) = SIN(a)/COS(a)
```

Si tenenos la tangente de un ángulo podemos conocer el valor del ángulo utilizando la función ATN (arco tangente). ATN es la función inversa de TAN:

```
a = ATN(0.6)
```

asigna a la variable a el valor del ángulo cuya tangente tiene un valor de 0,6. El ángulo se da en radianes.

El ordenador no conoce las inversas de las funciones COS y SIN, pero puedes definir tus propias funciones utilizando:

```
ARCSIN(a) = ATN(a/SQR(-a * a + 1))
ARCCOS(a) = ATN(a/SQR(-a * a + 1)) + PI/2
```

Otro número que se utiliza muy frecuentemente en matemáticas es el representado por la letra *e*:

Para obtener los catorce primeros dígitos de este número escribe:

```
PRINT EXP(1)
```

obtendrás:

```
2.7182818284588
```

La función **EXP** se define como:

```
EXP(X) = e^X
```

por lo que:

```
EXP(1) = e
```

La inversa de esta función es **LOG**. Esta función te da el logaritmo en base *e* del argumento, es decir, proporciona el logaritmo neperiano de un número que normalmente se escribe como ln.

Se relaciona con la función EXP de la siguiente forma:

EXP(X) = e^X = n
LOG(n) = X

Si prefieres utilizar logaritmos en base 10 utiliza la siguiente igualdad:

log 10(X) = LOG(X)/LOG(10)

Ejercicios

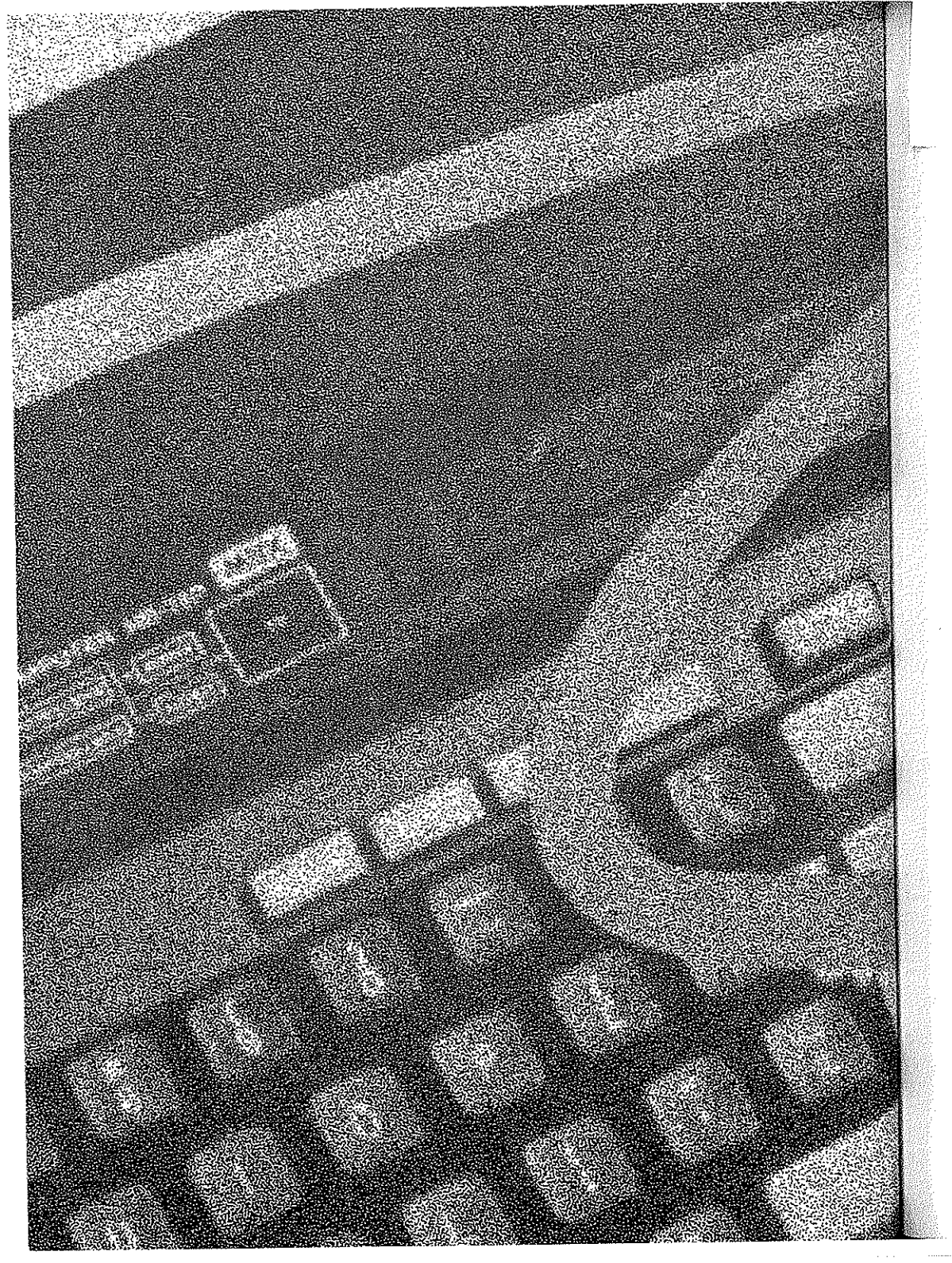
1. Define una función llamada PI, que te devuelva el valor del número PI; utiliza la siguiente igualdad:

   PI = ATN(1) * 4

2. Define una función que convierta grados en radianes. Utiliza la fórmula:

   a radianes = (a * PI)/180°
   Utiliza 3,14... para PI. Ahora prueba con la función PI.

3. Define una función que te dé el ARCCOS de su argumento. No olvides que $-1 <= COS(a) <= 1$, para un ángulo de "a" radianes. Es decir, la función sólo trabajará para argumentos entre +1 y −1 inclusive.
4. Define una función que devuelve el logaritmo en base 10 de su argumento.



20

El código ASCII

En el tema 6 descubrimos que las letras A-Z y a-z se encuentran almacenadas en memoria como los números 65-90 y 97-122, respectivamente. A estos números se les denomina "código ASCII".

ASCII significa *American Standard Code for Information Interchange* (código americano estándar para intercambio de información), y es el que generalmente utilizan los ordenadores para representar los caracteres.

Si quieres conocer el código ASCII de un carácter utiliza el comando ASC. Por ejemplo:

```
10 A1=ASC("A"): A2=ASC("a")
20 PRINT "El codigo ASCII de A es ";A1
30 PRINT "El codigo ASCII de a es ";A2
```

El argumento de la función ASC debe ser una cadena de caracteres. Si la cadena contiene más de un carácter, el ordenador devuelve el código ASCII del primero.

La función opuesta a ASC es CHR$. Cuando se proporciona un argumento numérico adecuado, CHR$ devuelve el correspondiente carácter ASCII a una variable del tipo cadena de caracteres. Por ejemplo:

```
10 A1$=CHR$(65): A2$=CHR$(97)
20 PRINT "El caracter correspondiente al codi
go ASCII 65 es ";A1$
30 PRINT "El caracter correspondiente al codi
go ASCII 97 es ";A2$
```

Los otros caracteres del teclado tienen también código ASCII. Busca los códigos ASCII para "%", "?" y "7".

De hecho hay 256 códigos ASCII, que van desde el 0 hasta el 255.

Puedes utilizar CHR$ para encontrar caracteres asociados con los códigos entre 32 y 255, inclusive. Ejecuta el siguiente programa para encontrar esos caracteres:

```
10 REM caracteres ASCII; 32-255
20 CLS
30 FOR I=32 TO 255
40 PRINT CHR$(I)+" ";
50 NEXT I
```

Obtendrás los caracteres alfanuméricos, así como la mayoría de los caracteres gráficos. Fíjate que el último carácter, que está representado por el código 255, es el cursor, y por ello cambia cuando mueves éste sobre la pantalla de texto.

Los códigos entre 0 y 31 son especiales.

A estos códigos se les denomina caracteres de control. En lugar de representar un carácter representan una determinada operación, que se debe ejecutar cuando se utiliza ese código como argumento en la instrucción PRINT CHR$.

Desde el tema 8 conoces el carácter de control 13. Ejecutando este código con PRINT CHR$ se hace equivalente a <RETURN>. Es la única forma que nos permite incluir <RETURN> en un programa:

```
10 REM algunos caracteres de control ASCII
20 PRINT CHR$(12)+"El caracter de control 12
limpia la  pantalla"
30 PRINT CHR$(7)+"El caracter de control 7 ha
ce un BEEP"
```

Los códigos 0 a 31 se utilizan también para representar el resto de los caracteres gráficos. Para acceder a ellos debes escribir en primer lugar PRINT CHR$(1) y luego CHR$(A+64), donde *A* es el número del código. El siguiente programa te escribe los treinta y dos primeros caracteres gráficos:


```
10 REM codigos 0-31 con caracteres graficos
20 FOR A=0 TO 31
30 PRINT CHR$(1)+CHR$(A+64)+" ";
40 NEXT A
```

La instrucción STRING$ se utiliza para escribir una cadena de caracteres de una determinada longitud y un carácter especificado. Esta cadena no puede exceder de 200 caracteres, a menos que hayas ejecutado anteriormente una instrucción CLEAR.

STRING$ debe ir seguido de dos argumentos, separados por comas e incluidos entre paréntesis.

El primer argumento debe ser un número, una expresión numérica o una variable numérica. Esta especifica la longitud de la cadena de caracteres.

El segundo argumento puede ser tanto un número como una cadena de caracteres. Si se utiliza un número, éste debe ser el código ASCII del carácter que debe ser escrito en la cadena de caracteres. Por ejemplo:

```
10 FOR B=1 TO 10
20 A$=STRING$(B,42)
30 PRINT A$
40 NEXT B
```

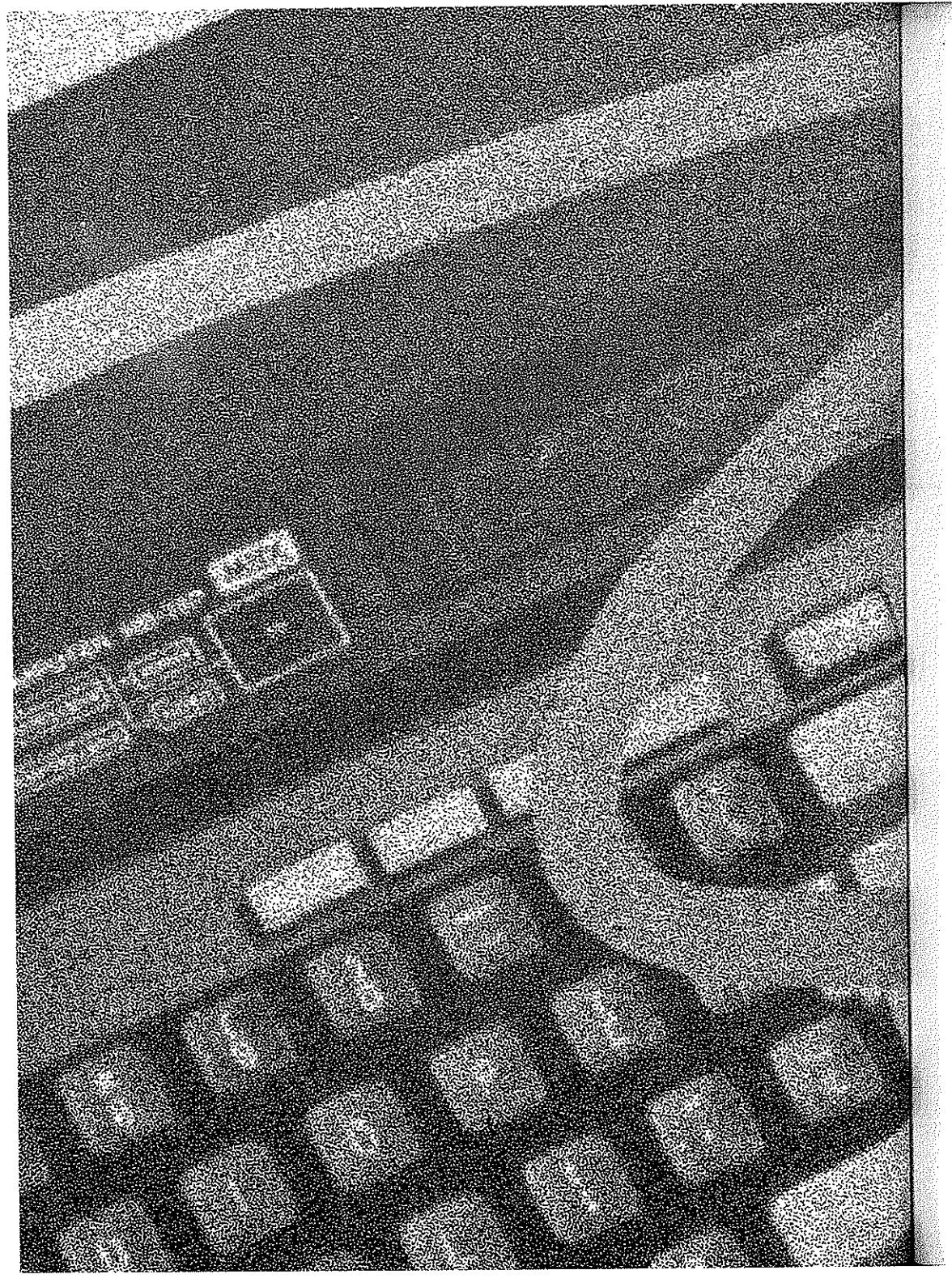
Obtendrás:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Si el segundo argumento es una cadena de caracteres, se escribe el primero de la cadena; es decir, la línea 20 tendría que modificarse a:

```
20 PRINT STRING$(B,"*")
```

y el efecto sería el mismo.



21

Modos de pantalla

Hasta el momento, únicamente hemos utilizado las pantallas de texto, SCREEN 0 y 1. Hay dos pantallas más, SCREEN 2 y 3. Las utilizaremos en los siguientes capítulos. Primero haremos un resumen de algunas de las características de cada pantalla.

Las pantallas de texto

Puedes cambiar la anchura de ambas pantallas utilizando la orden WIDTH (véase el tema 9):

PANTALLA	ANCHURA PREFIJADA	ANCHURA MAXIMA	TAMAÑO CAR
SCREEN 0	37	40	6 × 8
SCREEN 1	29	32	8 × 8

Las dos pantallas tienen veinticuatro líneas de texto. Aunque se puedan poner más caracteres en la pantalla 0 que en la 1, el tamaño de cada carácter es más pequeño. Unicamente 6 × 8 puntos para cada carácter en la pantalla 0, mientras que en la pantalla 1 hay 8 × 8 puntos por carácter. Esto significa que los caracteres alfanuméricos aparecen más comprimidos en la pantalla 0 que en la

pantalla 1. La compresión de la posición de los caracteres se percibe cuando se comparan caracteres gráficos.

El MSX es capaz de producir dieciséis colores. (Véase el tema 22 para más detalles.) En la pantalla 1 se pueden poner los colores: del texto, el fondo y el borde, con cualquier combinación de esos dieciséis colores. No puedes tener al mismo tiempo más de un color para el fondo. Los colores estándar que vienen ya prefijados son:

TEXTO	BLANCO
FONDO	AZUL OSCURO
BORDE	CIAN

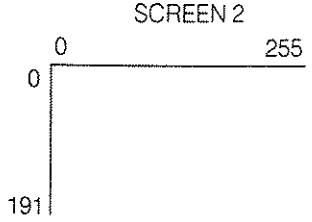
En la pantalla 0 sólo podrás fijar los colores del fondo y del texto. Los colores prefijados son idénticos a los de arriba.

Si en algún punto cambias el color de la tinta, todo el texto de la pantalla se cambiará a ese color.

Pantalla gráfica de alta resolución

La pantalla 2 (SCREEN 2) es la pantalla gráfica de alta resolución.

Tiene 256 puntos de anchura por 192 de largo. Para determinar un punto de esta pantalla se hace lo siguiente: el punto de la parte superior izquierda es la posición de coordenadas (0,0); el punto de la parte inferior derecha se corresponde a (255,191).



Aquí no podrás alterar la anchura de la pantalla. Si en la pantalla 2 tienes alguna instrucción WIDTH, el ordenador la memoriza y, cuando se pasa a la pantalla de TEXTO, fija la anchura especificada anteriormente. Si has escrito:

```
SCREEN 2
```

mientras estabas en el modo directo, te sorprenderás de que no haya ocurrido nada; esto se debe a que no puedes acceder a las pantallas gráficas desde el modo directo. La instrucción SCREEN la debes incluir en un programa para poder acceder a las pantallas gráficas, es decir:

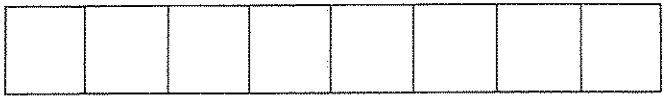

```
10 SCREEN 2
20 GOTO 20
```

La línea 20 congela la pantalla gráfica; esto es necesario, ya que el ordenador vuelve inmediatamente al modo texto en cuanto completa la ejecución de un programa. También volverá a la pantalla de texto cuando se encuentre una instrucción INPUT en un programa, o un error.

Para detener este programa y volver a la pantalla de texto presionar <CTRL><STOP>.

Los puntos en esta pantalla se ordenan en grupos de ocho puntos de longitud. El primer grupo de puntos se encuentra entre las coordenadas (0,0) y (8,0):

Un grupo de 8 × 1 puntos



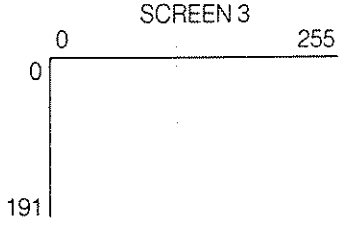
Cada grupo de ocho puntos sólo puede contener dos colores; el color del texto y el color del fondo. Grupos adyacentes pueden contener dos colores distintos. La resolución del color para la pantalla es de 32 × 192.

Para escribir caracteres en la pantalla consulta el tema 42, en la parte segunda. No puedes utilizar el comando PRINT de forma directa.

Pantalla multicolor

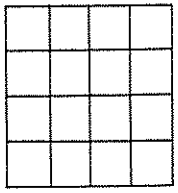
La pantalla 3 (SCREEN 3) es la pantalla multicolor, y puede utilizarse para gráficos de baja resolución.

Al igual que la pantalla 2, tiene 256 puntos de ancho por 192 de largo, pero no tiene una resolución de cada punto en pantalla.



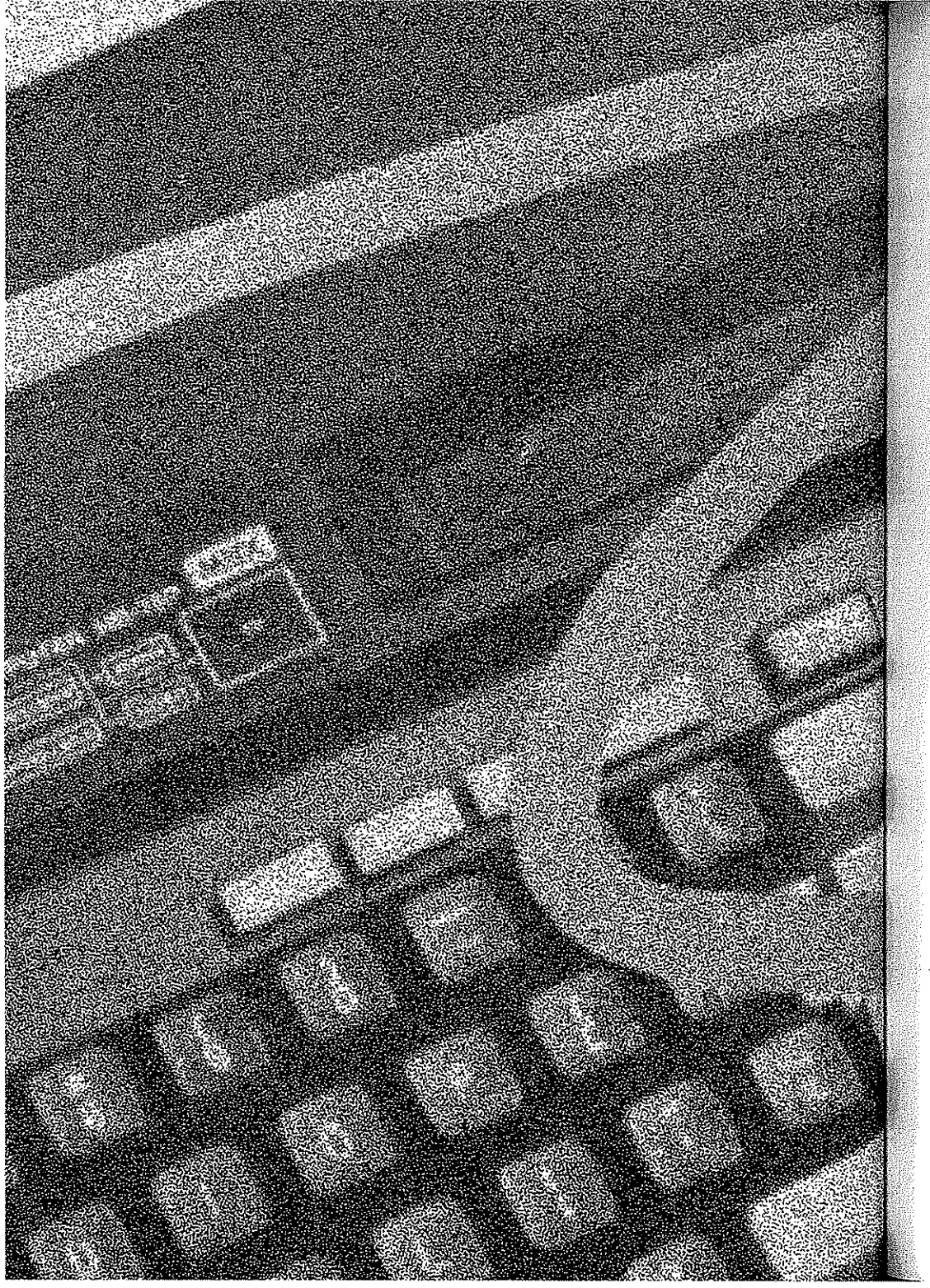
El punto más pequeño que puedes proyectar en esta pantalla es un bloque de 4 × 4 puntos. Puedes proyectar un punto (0,0), pero te encontrarás que aparece un bloque en pantalla y también cubre los puntos (1,0), (2,0), (3,0) hasta (3,3). Proyectar (3,3) tendría el mismo efecto.

Un bloque de 4 × 4 puntos:



Cada bloque 4 × 4 sólo puede contener un color: el del texto. Por tanto, la resolución del color de esta pantalla es de 64 × 48.

Como en la pantalla 2 no puedes acceder a esta pantalla desde el modo directo, deberás congelar la pantalla utilizando un bucle infinito. Un comando INPUT o un error te devolverán a la pantalla de texto.



22
Color

Al conectar el ordenador, los colores prefijados para la pantalla son: el azul oscuro, para el fondo; el borde, en color cian, y, el texto, en blanco.

Puedes cambiar uno o todos los colores utilizando el comando COLOR.

A continuación de recibir el comando COLOR el ordenador espera tres números separados por comas. El primer número fija el color del texto; el segundo, el fondo, y el tercero, el borde. Estos son los dieciséis colores que puedes utilizar:

0 Transparente
1 Negro
2 Verde
3 Verde (claro)
4 Azul (oscuro)
5 Azul (claro)
6 Rojo (oscuro)
7 Cian
8 Rojo (medio)
9 Rojo (claro)
10 Amarillo (oscuro)
11 Amarillo (claro)
12 Verde (oscuro)
13 Magenta
14 Gris
15 Blanco

o sea, el color estándar es:

COLOR 15,4,7

La tecla de función F6 está programada para ejecutar COLOR 15,4,7, seguida de <RETURN>. Luego si accidentalmente has fijado el color de la tinta en la pantalla pulsa esta tecla y volverás a los colores iniciales.

La tecla F1 está programada para escribir en la pantalla COLOR. No tienes más que escribir los tres números para los colores, pudiendo dar por separado el número del color que quieres cambiar. Por ejemplo:

COLOR 1

Fija el texto en negro, dejando el borde y el fondo tal como estaban.

COLOR,15,15

Se cambian los colores del fondo y del borde a blanco, dejando el texto en negro. Fíjate que aunque no se haya especificado el color del texto, sí debe ponerse la coma que va después, ya que esto indica que el siguiente color fija el fondo.

COLOR,,10

Fija el borde en amarillo oscuro. Para poder advertirlo tienes que estar en las pantallas 1, 2 ó 3, puesto que la pantalla 0 no tiene borde.

Si cambias el color del fondo mientras te encuentras en las pantallas gráficas deberás ejecutar el comando CLS para poder observar el efecto, ya que se ha de borrar la pantalla, para dar paso al nuevo color de fondo.

El siguiente programa visualiza las combinaciones de colores del borde y del fondo permitidas en el MSX. Escríbelo en SCREEN 1:

```
10  REM Colores
20  FOR BRD=0 TO 15
30  COLOR,,BRD
40  FOR PAP=0 TO 15
50  COLOR ,PAP
60  FOR RETARDO=1 TO 200:NEXT RETARDO
70  NEXT PAP
80  NEXT BRD
90  COLOR 15,4,7
```

El bucle de la línea 60 te proporciona el tiempo para que puedas ver las diferentes combinaciones de colores; sin esta línea pasarían todos los colores tan deprisa que no te daría tiempo a verlos.

Puedes utilizar variables y expresiones numéricas en una instrucción COLOR. Si utilizas un número real en esta instrucción se truncará el entero más próximo. El número debe estar dentro del rango de 0 a 15.

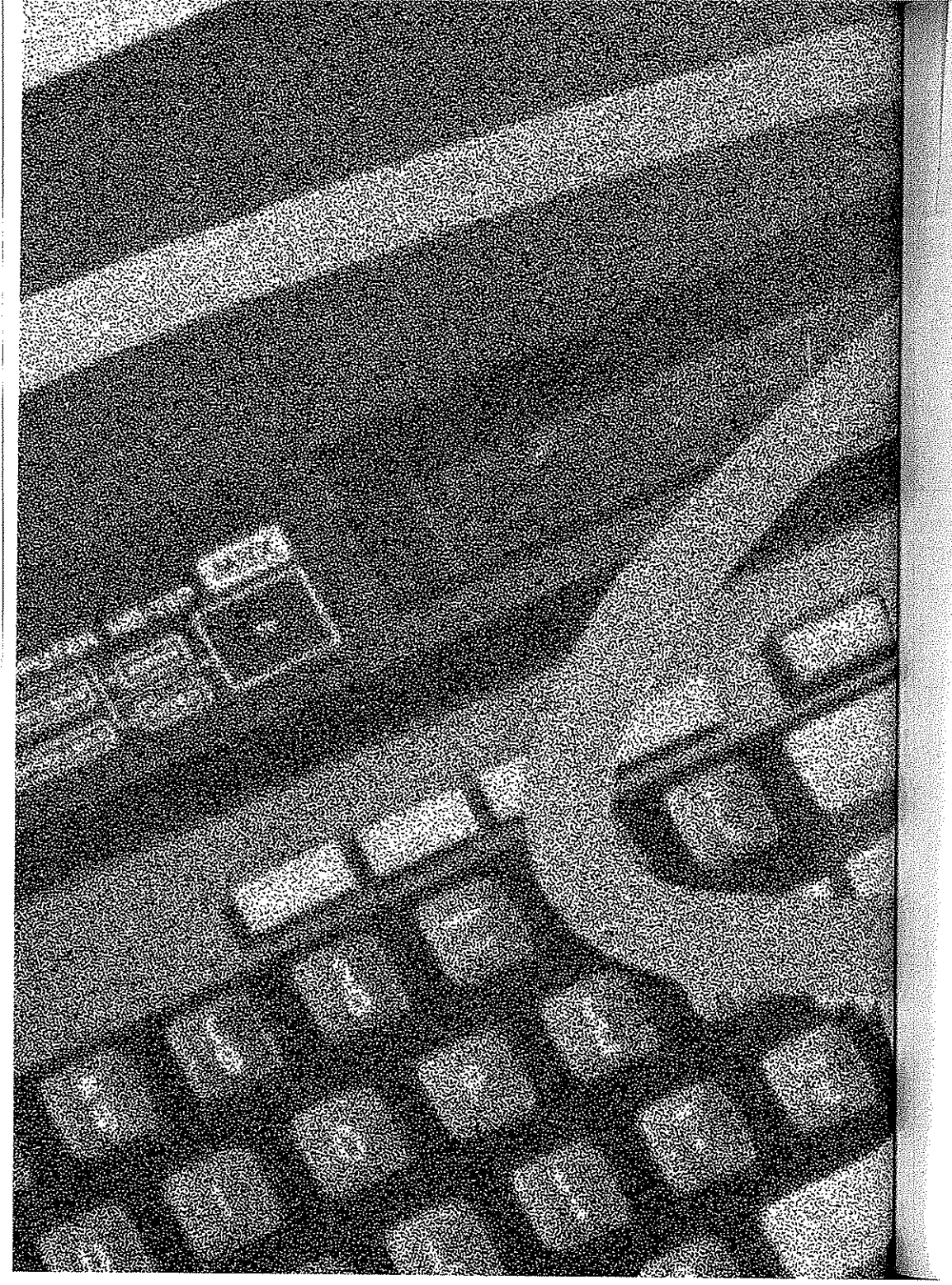
Intenta ejecutar este programa en las otras pantallas. Necesitarás insertar el comando CLS para las pantallas 2 y 3.

¿Te has dado cuenta de lo que hace COLOR 0?

Fija el texto y el fondo al mismo color que el borde. Si se utiliza para fijar el color del borde, éste se vuelve negro.

Ejercicio

1. Programa una de las teclas de función para poder ejecutar diferentes combinaciones del color de texto, fondo y borde.



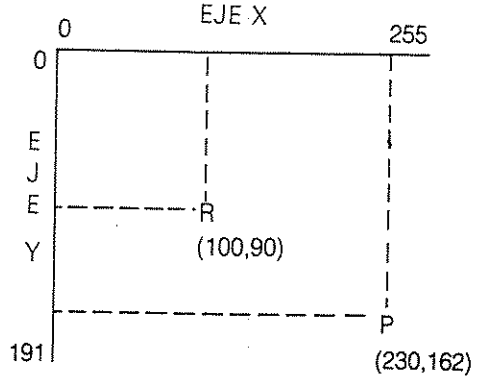
23
Dibujar puntos

Los primeros comandos que se utilizan en este capítulo son PSET y PRESET, y ambos son muy similares. Al ser comandos gráficos, sólo pueden utilizarse en las pantallas 2 y 3.

PRESET se utiliza para fijar un punto en el color del fondo. Las coordenadas del punto pueden ser absolutas o relativas.

A continuación de utilizar PRESET observarás que el color del texto es el mismo que el del fondo.

En ambas pantallas, 2 y 3, la coordenada X debe encontrarse dentro del rango comprendido entre 0 y 255, y la coordenada Y en el comprendido entre 0 y 191. Las coordenadas absolutas se leen directamente de los ejes X e Y:



El punto R tiene como coordenadas absolutas: 100 para la coordenada X y 90 para la coordenada Y. P tiene como coordenadas absolutas (230,162).

La coordenada relativa proporciona la posición de un punto respecto de otro de referencia. Por ejemplo, el punto P tiene como coordenadas relativas (130,72), medidas respecto del punto R. Esto significa que P se encuentra 130 puntos más a la derecha que R sobre el eje X, y 72 más abajo en el eje Y.

En todas las instrucciones gráficas el punto de referencia que se toma siempre es la última posición del cursor gráfico.

La posición del cursor gráfico puede fijarse con PRESET:

```
10 SCREEN 2
20 PRESET (100,100)
30 GOTO 30
```

Cuando ejecutes este programa pensarás que no ha ocurrido nada. Sin embargo, el cursor gráfico se encuentra ahora posicionado en el punto (100,100). Cualquier comando gráfico que utiliza coordenadas relativas inmediatamente después de estas líneas utilizará el punto (100,100) como punto de referencia.

Si quieres dibujar una línea deberás especificar el color de ésta, o cambiar primero el color del texto con la instrucción COLOR; en caso contrario tu línea será invisible.

Si especificas un color en la instrucción PRESET, se trazará un punto de ese color, y el color del fondo se convertirá a éste. Sustituye la línea 20 por:

```
20 PRESET (100,100),1
```

El número que sigue a la instrucción PRESET debe ser uno de los códigos de color especificados en el tema 22; es decir, un número comprendido entre 0 y 15. Si se utiliza un número real comprendido dentro de este rango se ignora la parte decimal.

Si las coordenadas sobrepasan la pantalla, entonces no verás nada. Si las coordenadas sobrepasan el rango entero permitido (−32768,32767) se producirá un mensaje de error.

El comando **PSET** traza un punto del color del texto en las coordenadas especificadas. Estas coordenadas pueden ser absolutas o relativas.

Si se especifica un color en la instrucción, entonces se traza un punto de este color; es idéntico a utilizar PRESET con un color especificado y, al igual que PRESET, esta instrucción también cambia el color del texto al color especificado.

El siguiente programa traza bloques de colores, aleatoriamente, en la pantalla 3, utilizando el comando PSET. Estos bloques se borrarán al poner los mismos puntos aleatorios en el color del fondo utilizando PRESET:

```
10   REM Puntos
20   R=TIME
30   RDRST=RND(-R)
40   SCREEN 3
50   REM Puntos de colores aleatorios
60   FOR I%=1 TO 300
70   GOSUB 1000
80   PSET (X%,Y%),Z%
90   NEXT I%
100  REM Borrar los puntos
110  RDRST=RND(-R)
120  FOR J%=1 TO 300
130  GOSUB 1000
140  PRESET(X%,Y%)
150  NEXT J%
160  END
990  REM Subrutina aleatoria
1000 X%=RND(1)*256
1010 Y%=RND(1)*192
1020 Z%=RND(1)*16
1030 RETURN
```

Este programa utiliza variables enteras al objeto de acelerar la ejecución, es decir, se utilizará X% en lugar de X.

Cada vez que se ejecuta el programa los puntos se trazan en distintas posiciones aleatorias. Esto es debido a que el punto de la secuencia de números aleatorios se fija a partir de una variable TIME, y esta variable tendrá distintos valores cada vez que se ejecute el programa. Tanto en la primera como en la segunda parte del programa se generarán los mismos números aleatorios, borrándose todos los puntos. Para generar los mismos números la secuencia de números aleatorios se inicializa utilizando el mismo argumento negativo en la función RND de la línea 110, tal como lo utilizamos previamente en la línea 30.

Finalmente, el comando **POINT**, que es otro comando gráfico, devuelve el código de color de un punto en cualquiera de las pantallas gráficas. El número devuelto está comprendido entre 0 y 15, representando los colores que van desde el transparente al blanco. Se devuelve −1 si las coordenadas de la instrucción POINT sobrepasan la pantalla. Escribe:

```
10 COLOR ,2
20 SCREEN 2
30 P=POINT(100,100)
```

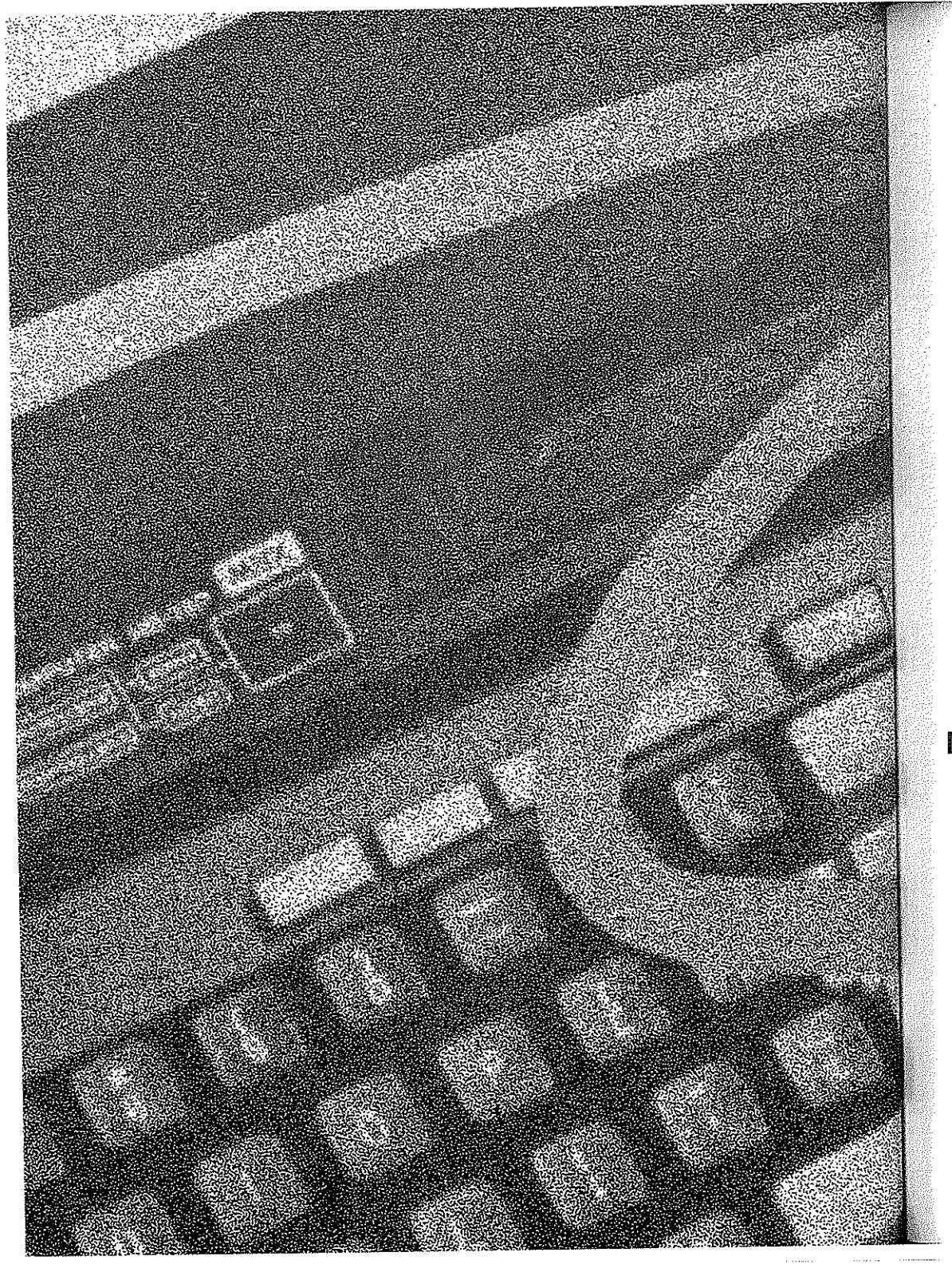
Ahora, volviendo al modo directo, escribe:

```
PRINT P
```

Se visualizará el número 2, ya que es el código del color verde pálido, que es el nuevo color de fondo.

Ejercicio

1. Intenta trazar la curva del SIN, utilizando:
   Y = A*SIN(X) + A
   A fija la amplitud de la curva. Tienes que ir variando X dentro del rango comprendido entre 0 y 2*PI.



24

Dibuja rectas y rectángulos

Este capítulo trata del comando LINE, otro de los comandos gráficos, que sólo puede utilizarse en las pantallas 2 y 3.

Dibujando líneas

Cuando dibujes una línea deberás especificar el punto inicial y final, que podrán darse tanto en coordenadas absolutas como en relativas, o incluso como mezcla de las dos. Escribe el siguiente programa:

```
10 SCREEN 2
20 LINE(0,0)-(255,191)
30 GOTO 30
```

Este programa dibujar una diagonal a través de la pantalla. Fíjate en el signo "-" que se encuentra entre las dos coordenadas.

Se han utilizado coordenadas absolutas. El primer par de coordenadas (0,0) proporciona la posición inicial de la línea, mientras que la coordenada (265,191) proporciona la posición final.

Cambia la línea 10 del programa para ejecutarlo en el modo 3 y comparar la resolución de las dos pantallas gráficas.

Prueba a utilizar distintas coordenadas para el comando LINE.

Si las coordenadas X e Y son mayores que 255 y 191, respectivamente, el ordenador dibuja hasta el final de la pantalla, tomando X como 255 e Y como 191. Si tus coordenadas sobrepasan el rango entero permitido, de −32768 hasta 32767, se producirá un mensaje de error.

No necesitas especificar el punto inicial de LINE. De ser así el ordenador comenzará a dibujar a partir de la posición en que se encuentre el cursor gráfico. Si acabas de ejecutar un LINE, el cursor se encontrará en la posición final de la línea trazada.

Puedes utilizar coordenadas relativas para dibujar la misma línea. Sustituye la línea 20 del programa por:

```
20 LINE(0,0)-STEP(255,191)
```

La instrucción STEP le dice al ordenador que las siguientes coordenadas son relativas. Por lo que, en este caso, la posición final de X se encontrará 255 puntos más a la derecha del eje X, y la posición final de Y se encontrará 191 puntos más abajo en el eje Y, situando el último punto de LINE en las coordenadas absolutas (255,191).

Líneas coloreadas

Hasta ahora sólo has dibujado líneas con el color del texto. Es posible especificar el color de la línea poniendo una coma, y un código de color válido, a continuación del comando LINE; es decir, poniendo como código un número entre 0 y 15. (Véase el tema 22.) Ahora sustituye la línea 20 del último programa por

```
20 LINE -(255,191),10
```

Esto dibuja una línea diagonal amarilla a través de la pantalla.

El siguiente programa te muestra la resolución de color en la pantalla 2:

```
10 SCREEN 2
20 FOR A=0 TO 15
30 LINE(0+A,0)-(200+A,191),A
40 NEXT A
50 GOTO 50
```

Verás dieciséis líneas diagonales a través de la pantalla, pero sus colores son bastante borrosos. Si ahora sustituyes la línea 30 por:

```
30 LINE(0,0+A)-(255,0+A),A
```

obtendrás la resolución del color de cada línea.

Obtienes estos efectos por la forma en que se almacena la pantalla en memoria del ordenador. Cada línea horizontal se divide en grupos de ocho puntos de longitud; cada grupo sólo puede tener dos colores de una sola vez, un color para el fondo y el otro para el texto. Cuando se traza la primera línea diagonal el color del texto se fija en 0 y la línea se dibuja en ese color. La siguiente línea fija el color del texto a 1; esto cambia el color del texto para el grupo completo de ocho puntos, por lo que el primer punto de la primera línea cambia su color al negro. La octava línea trazada cambia el color del texto del primer grupo de ocho puntos por última vez (este grupo completo es ahora cian).

La resolución de color vertical es mejor que la horizontal. El color de cada punto vertical es independiente de los puntos de arriba y abajo; por tanto, las dieciséis líneas horizontales no interfieren una con la otra.

Intenta ejecutar los dos programas en modo 3.

En esta pantalla verás las mismas cuatro líneas cada vez, ya que la resolución de color vertical y horizontal es la misma. Puedes tener un color por cada bloque de 4 × 4 puntos. La resolución de color es la misma que la resolución gráfica. Las primeras cuatro líneas se trazan unas sobre otras, por lo que sólo ves la última, que es verde claro.

Dibujando cuadrados

Es posible dibujar un cuadrado utilizando cuatro comandos LINE, uno a continuación del otro. Sin embargo, hay una forma más sencilla; conociendo las coordenadas de las esquinas superior izquierda e inferior derecha. Continuarás utilizando el comando LINE, pero ahora el primer par de coordenadas dan la esquina superior izquierda y el segundo par serán las coordenadas de la esquina inferior derecha. Para indicarle al ordenador que deseas un cuadrado y no una línea, debes poner el código de color seguido por una coma y una letra "B". Escribe el siguiente programa:

```
10 REM cuadrados
20 SCREEN 2
30 FOR A=0 TO 90 STEP 10
40 LINE(5+A,5+A)-(255-A,185-A),A/10,B
50 NEXT A
60 GOTO 60
```

Este programa te dibujará diez cuadrados de diferentes tamaños y colores. Fíjate que puedes utilizar una expresión para evaluar el código de color, pero esta expresión debe proporcionar un número entre 0 y 15. Si el resultado fuese un número real, se truncará la parte decimal.

Ahora ejecuta este programa en la pantalla 3; cambia la línea 20 por:

```
20 SCREEN 3
```

Si conoces las longitudes de los lados de los cuadrados, en lugar de las coordenadas de las esquinas será más fácil utilizar coordenadas relativas para el segundo par de coordenadas, es decir, dibujar un cuadrado de 100 puntos de longitud en la dirección X, y 50 puntos de anchura en la dirección Y, puedes utilizar:

```
10 SCREEN 2
20 LINE(10,10)-STEP(100,50),,B
30 GOTO 30
```

El programa dibuja un cuadrado con las dimensiones dadas y en el color actual del texto. La esquina superior derecha está situada en coordenadas absolutas (10,10).

El segundo par de coordenadas especifica las longitudes de los lados del cuadrado, que son paralelos a los ejes X e Y.

Este es el mejor método para dibujar cuadrados. Por ejemplo:

```
10 REM cuadrados
20 SCREEN 2
30 FOR A=10 TO 90
40 LINE(A,A)-STEP(A,A),,B
50 NEXT A
60 GOTO 60
```

Este programa dibuja cuadrados de tamaños crecientes en diagonal a través de la pantalla.

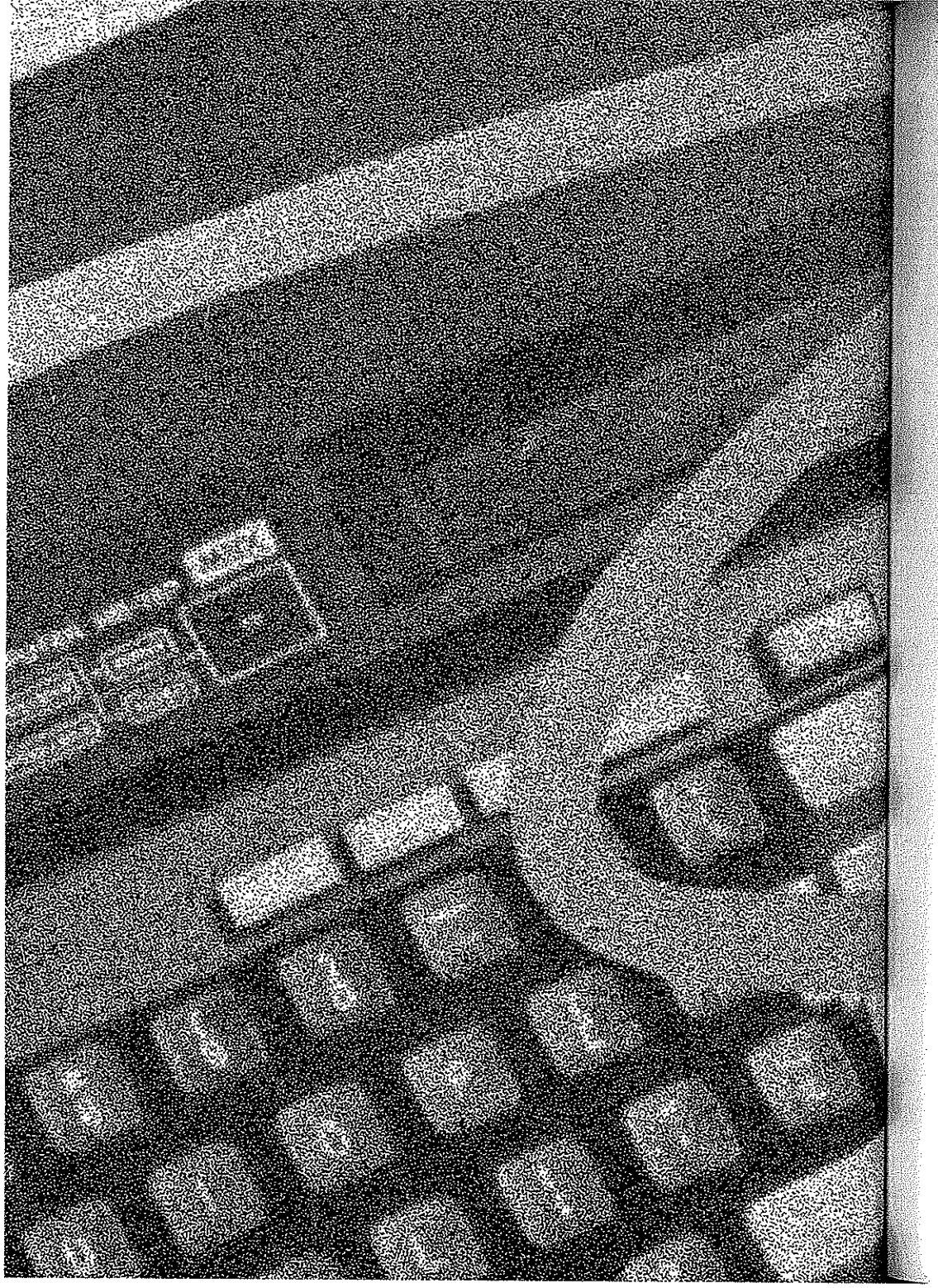
Cuadrados coloreados

Para dibujar un cuadrado y pintarlo de un determinado color sustituye la letra "B" por "BF". El color que se especifica en la instrucción LINE se utiliza ahora para pintarlo:

```
10 REM cuadrado negro
20 SCREEN 2
30 LINE(20,20)-(100,100),1,BF
40 GOTO 40
```

Ejercicio

1. Escribe un programa que dibuje un cuadrado de tamaño aleatorio en una posición de pantalla aleatoria. Pinta el cuadrado con un color aleatorio.



25

Círculos y elipses

El comando CIRCLE permite dibujar círculos y elipses, tanto enteros como en parte. Debes especificar la posición del centro del círculo y para ello puedes utilizar tanto coordenadas absolutas como relativas y la longitud del radio; también puedes indicar el color del círculo y la parte de éste que quieres dibujar. CIRCLE es un comando gráfico y, por consiguiente, sólo puede utilizarse en las pantallas gráficas.

Círculos

Para dibujar un círculo debes especificar las coordenadas de su centro, así como la longitud del radio. Por ejemplo:

```
10 REM circulos
20 SCREEN 2
30 CIRCLE(128,100),90
40 GOTO 40
```

Dibuja un círculo con centro de coordenadas absolutas (128,100), y un radio de 90 puntos, con el color de tinta actual. Si especificas un radio demasiado grande para la pantalla no se verá el círculo o sólo parte de él.

Para especificar un color para el círculo pon una coma después del radio y un código de color válido, es decir, un número que se encuentre comprendido entre 0 y 15. (Véase el tema 22 para los códigos de color.) El siguiente programa dibuja el mismo círculo en negro:

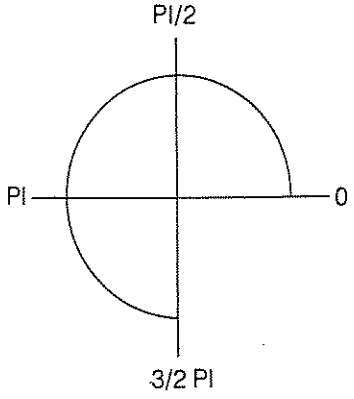
```
10 SCREEN 2
20 CIRCLE(128,100),90,1
30 GOTO 30
```

Dibujo de arcos

Para dibujar parte de un círculo, o un arco, se debe especificar el ángulo inicial y el final. El siguiente programa dibuja tres cuartos de círculo en negro:

```
10 REM arcos
20 SCREEN 2
30 PI=4*ATN(1)
40 CIRCLE(128,100),90,1,0,3*PI/2
50 GOTO 50
```

Los dos últimos números que se han añadido a la instrucción CIRCLE son los ángulos inicial y final del arco. Los ángulos deben ir especificados en radianes. (Si no recuerdas lo que es un radián repasa el tema 19.) La línea 30 sólo calcula el valor de PI y el arco se dibuja entre 0 y 3*PI/2.



Todos los ángulos se miden a partir del eje X, en sentido opuesto a las agujas del reloj. Como hay 2*PI radianes en un círculo, los ángulos especificados pueden estar comprendidos entre 0 y 2*PI.

Arcos delimitados por su radio

Para hacerlo pon el signo "−" antes de los ángulos inicial y final. Esto hará que se dibujen unas rectas desde el centro del círculo hasta las posiciones inicial y final del arco. Ejecuta el siguiente programa:

```
10 REM dibujando arcos delimitados
20 SCREEN 2
30 PI=4*ATN(1)
40 CIRCLE(128,100),90,,-PI/2,-PI
50 GOTO 50
```

Verás el sector superior izquierdo del círculo con el color actual del texto.

Elipses

Para dibujar una elipse debes especificar la variación del radio. Esta se define como:

$$\text{VARIACION DEL RADIO} = \frac{\text{RADIO VERTICAL}}{\text{RADIO HORIZONTAL}}$$

Si la variación es mayor que 1, entonces la elipse se alargará verticalmente. El radio especificado se toma como el radio vertical.

Si la variación es menor que 1, entonces la elipse se alargará horizontalmente. Y el radio especificado se toma como el radio horizontal.

El siguiente programa dibuja una elipse completa, en color negro:

```
10 REM elipses
20 SCREEN 2
30 CIRCLE(128,100),90,1,,,1.5
40 GOTO 40
```

Como la variación del radio es de 1,5, la elipse se alarga verticalmente y el radio vertical es de 90 puntos.

Sustituyendo la línea 30 por:

```
30 CIRCLE(128,100),90,1,,,.5
```

Verás una elipse alargada horizontalmente, con un radio horizontal de 90 puntos.

Fija la variación del radio a 100 y 0,01; las elipses aparecerán como líneas vertical y horizontal, respectivamente, con una longitud de 180 puntos.

El siguiente programa dibuja elipses coloreadas, con variaciones del radio aleatorias; todas se sitúan en el centro de la pantalla:

```
10 REM muestras de elipses
20 SCREEN 2
30 FOR B=1 TO 15
40 A=RND(-TIME)*2
50 CIRCLE(128,96),90,B,,,A
60 NEXT B
70 GOTO 70
```

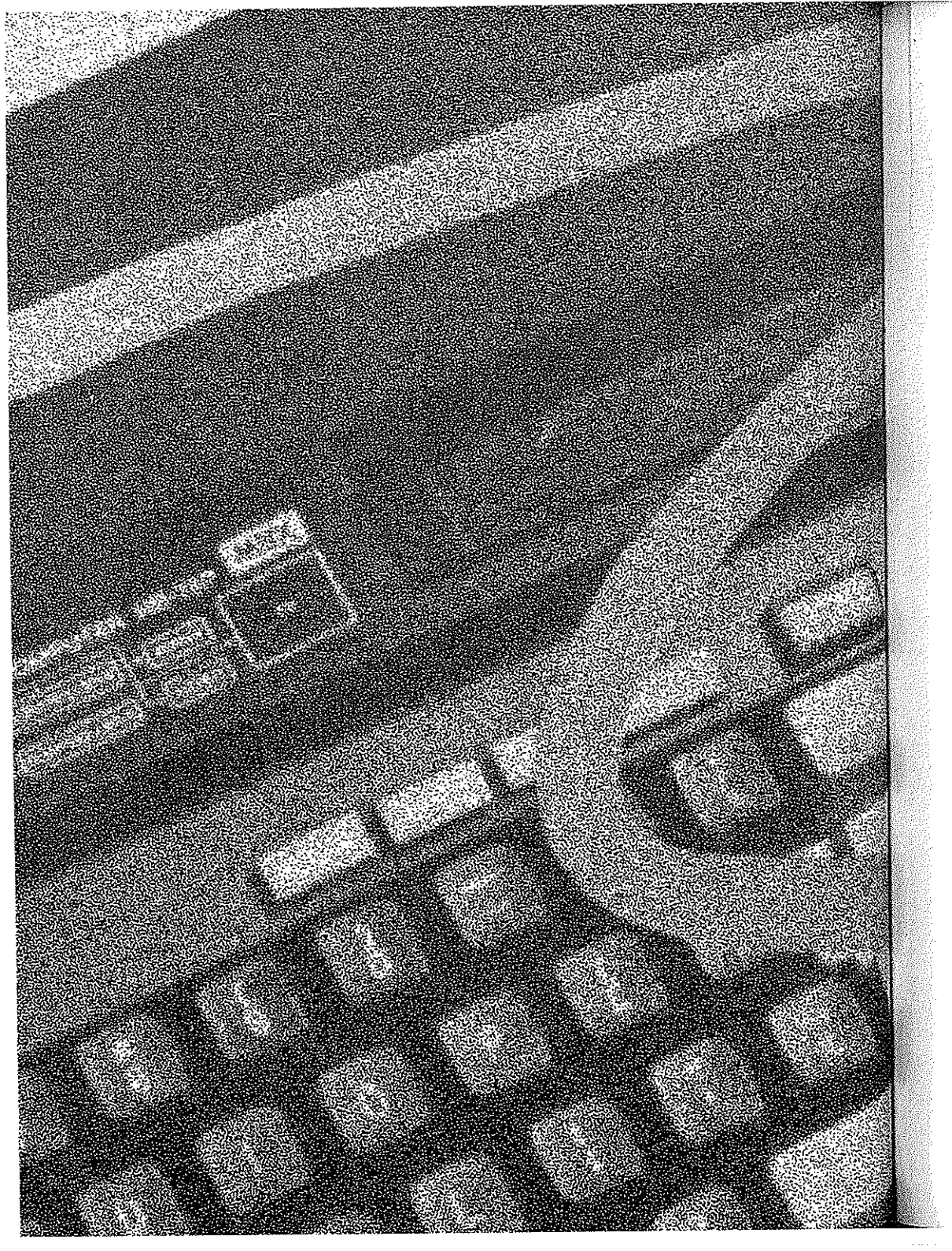
Si quieres dibujar una parte de la elipse debes especificar los ángulos inicial y final, tal como hiciste cuando dibujabas arcos. También puedes emplear el signo "_":

```
10 SCREEN 3
20 PI=4*ATN(1)
30 FOR B=1 TO 15
40 A=B/10
50 SA=RND(-TIME)*2*PI
60 EA=RND(-TIME)*2*PI
70 CIRCLE(126,96),90,B,SA,EA,A
80 NEXT B
90 GOTO 90
```

Este programa dibuja formas muy curiosas, centradas en el punto (126,96). Los ángulos inicial y final se calculan utilizando la función RND. Fíjate que el ángulo final puede ser menor que el inicial. La variación del radio varía entre 0,1 y 1,5. Ejecuta el programa en la pantalla 2 al objeto de comparar la resolución.

Ejercicios

1. Escribe un programa que dibuje diez pequeños círculos o elipses en posiciones aleatorias de la pantalla.
2. Cambia el programa para que los círculos tengan distintos radios.
3. Dibuja un círculo que conste de cuatro sectores; cada sector debe dibujarse en un color distinto.



26

El macrolenguaje gráfico

El macrolenguaje gráfico aparenta el movimiento de un lapicero sobre un papel y permite dibujar figuras complicadas de una forma muy sencilla. Utilizando el comando DRAW puedes mover el cursor gráfico a cualquier punto de la pantalla.

El comando DRAW tiene que ir seguido por una cadena de caracteres. Esta cadena contiene los macrocomandos para gráficos, representados por un único carácter.

Para dibujar una línea en una de las cuatro direcciones, es decir, arriba, abajo, derecha o izquierda, puedes utilizar cualquiera de los siguientes comandos gráficos:

U Dibuja hacia arriba.
D Dibuja hacia abajo.
R Dibuja hacia la derecha.
L Dibuja hacia la izquierda.

Debes especificar la longitud de la línea a continuación del comando. Esta longitud se mide en puntos.

El siguiente programa te muestra cómo utilizar el macrolenguaje gráfico:

```
10 REM cuadrado
20 SCREEN 2
```

```
30 PSET(123,91)
40 DRAW"R10D10L10U10"
50 GOTO 50
```

El ordenador ejecuta los macrocomandos secuencialmente, es decir, dibuja una línea a la derecha con una longitud de 10 puntos; luego, otra hacia abajo de 10 puntos; a continuación, a la izquierda de 10 puntos, y finalmente hacia arriba, también de 10 puntos, completando así el cuadrado.

Utilizando los cuatro comandos siguientes podrás dibujar líneas diagonalmente:

E Dibuja una diagonal hacia arriba a la derecha.
F Dibuja una diagonal hacia abajo a la derecha.
G Dibuja una diagonal hacia abajo a la izquierda.
H Dibuja una diagonal hacia arriba a la izquierda.

Cada uno de estos comandos debe ir seguido por un número que especifica el número de puntos que deben recorrerse en las direcciones X e Y.

Por ejemplo, las coordenadas del último punto de la línea dibujada por E 10 sería (10,10), tomando el primer punto de esa línea como referencia.

El siguiente programa dibuja un hexágono:

```
10 REM hexagono
20 SCREEN 2
30 PSET(128,96)
40 DRAW"E20F20D30G20H20U30"
50 GOTO 50
```

Para dibujar una línea hasta un determinado punto de la pantalla utiliza el comando M. Este comando debe ir seguido por unos valores para las coordenadas X e Y del último punto de la línea que quieres trazar. Los valores asignados a X e Y pueden ser absolutos o relativos. Por ejemplo:

M40,50

mueve el cursor gráfico a la posición de coordenadas absolutas (40,50), mientras dibuja una línea.

Para especificar coordenadas relativas emplea los prefijos + o −:

M+40,50 Dibuja hasta una posición +40 puntos a lo largo del eje X y +50 a lo largo del eje Y.
M+40,−50 Dibuja hasta una posición +40 puntos a lo largo del eje X y −50 a lo largo del eje Y.

Los siguientes comandos son equivalentes:

M+0,−10=U10	M+10,−10=E10
M+0, 10=D10	M+10,10=F10
M+10,0=R10	M−10,10=G10
M−10,0=L10	M−10,−10=H10

En el último programa se utiliza el comando PSET para posicionar el cursor gráfico, antes de dibujar. También se puede hacer mediante el prefijo B antes del comando M. B significa "blanco", y provoca que el cursor se mueva sin dibujar.

30 DRAW"BM128,96"

puede utilizarse en un programa para posicionar el cursor en el centro de la pantalla.

El prefijo B también puede utilizarse con los macrocomandos que ya conoces; estos comandos son casos particulares del comando M.

Si después de haber dibujado una línea quieres volver a la posición inicial utiliza el prefijo N.

El siguiente programa dibuja una estrella en la pantalla:

```
10 REM estrella
20 SCREEN 2
30 REM mover cursor
40 DRAW"BM128,96"
50 REM dibujar los brazos
60 DRAW"NU50ND50NR50NL50"
70 DRAW"NE30NF30NG30NH30"
80 GOTO 80
```

Cada vez que se dibuja un brazo el cursor vuelve al centro de la estrella.

Ahora ya tienes que ser capaz de dibujar figuras complicadas. Una vez que has definido tu figura es posible cambiar su orientación en la pantalla utilizando el comando A.

Este comando rota los ejes en la pantalla, en la dirección opuesta a la de las agujas del reloj, en 0, 90, 180 ó 270 grados, a partir de su orientación normal. Esto afecta a la orientación de los comandos de dirección U, D, L, R, F, E, G y H.

El comando A debe ir seguido por un número entero entre 0 y 3, donde:

An Rotación.
A0 Fija los ejes en la orientación estándar.
A1 Rota los ejes en la dirección contraria a la de las agujas del reloj 90 grados.

A2 Rota los ejes en la dirección contraria a la de las agujas del reloj 180 grados.

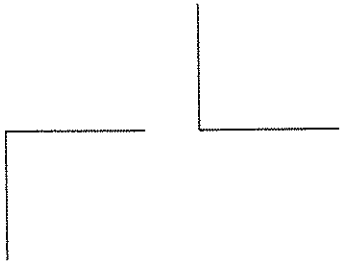
A3 Rota los ejes en la dirección contraria a la de las agujas del reloj 270 grados.

Una vez que se ha ejecutado A, la orientación de todos los otros comandos de dirección del programa se rotarán en el ángulo fijado. Para volver a la posición inicial utiliza A0 dentro del comando DRAW.

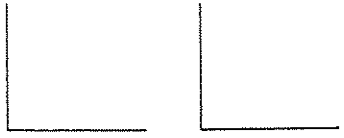
Ejecuta dos veces el siguiente programa:

```
10 REM angulo
20 SCREEN 2
30 DRAW"BM50,100NR50ND50"
40 DRAW"A1BM150,100NR50ND50"
50 GOTO 50
```

La primera vez que ejecutes el programa verás:



y la segunda:



Esto ocurre porque el comando A tiene aún el valor A1. Si quieres repetir el primer resultado deberás inicializar el ángulo de rotación a A0 al principio de la cadena de caracteres de la línea 30.

Hasta ahora todos los dibujos tienen idéntico color que el texto. Para dibujar en un color distinto, es decir, para cambiar el color del texto, utiliza el comando C, que debe ir seguido por un código de color válido. Los códigos de color son los mismos que para el comando COLOR del tema 22. Por lo que:

C0 Transparente
C1 Negro
C2 Verde pálido
C3 Verde claro
C4 Azul oscuro
C5 Azul claro
C6 Rojo oscuro
C7 Cian
C8 Rojo intermedio
C9 Rojo claro
C10 Amarillo oscuro
C11 Amarillo claro
C12 Verde oscuro
C13 Magenta
C14 Gris
C15 Blanco

Al igual que en el comando A, una vez que se haya fijado C el texto permanecerá con ese color hasta que se cambie de nuevo con los comandos C o COLOR.

El comando S cambia la escala o tamaño del dibujo. El número que sigue a S fija el factor de escala. Este número puede ser cualquier entero comprendido entre 0 y 255.

El factor de escala (SF) se define como:

$$SF = n/4$$

S1 proporciona un SF de 1/4. Todas las longitudes especificadas por los comandos U, D, L, R, etc., se reducen a una escala de 1/4. Ejecuta el siguiente programa:

```
10 REM escala
20 SCREEN 2
30 DRAW"BM50,50"
40 DRAW"R150D100L150U100"
50 DRAW"BM50,50"
60 DRAW"S1R150D100L150U100"
70 GOTO 70
```

Verás dos rectángulos, uno de ellos una cuarta parte menor que el otro. Si ejecutas de nuevo el programa, ambos rectángulos se dibujarán uno sobre el otro, con el tamaño reducido.

Para inicializar SF puedes utilizar S4 o S0. Inserta uno de los dos al principio de la cadena de caracteres de la línea 40 y ejecuta de nuevo el programa.

Si quieres dibujar la misma figura más de una vez puedes tener en tu programa una variable de tipo cadena de caracteres, asignándole la cadena que contenga todos los macrocomandos necesarios. El siguiente programa dibuja en pantalla cuatro diamantes:

```
10  REM Diamantes
20  SCREEN 2
30  A$="F40G40H40E40"
40  DRAW"BM50,8"
```

```
50  DRAW A$
60  DRAW"BM200,8"
70  DRAW A$
80  DRAW"BM50,104"
90  DRAW A$
100 DRAW"BM200,104"
110 DRAW A$
120 GOTO 120
```

El programa se reduce bastante si pones más de un comando en cada sentencia DRAW. Para ejecutar el contenido de una variable del tipo cadena de caracteres dentro de DRAW debes utilizar el comando X. El siguiente programa también dibuja cuatro diamantes, pero esta vez son de diferente tamaño. La variable de tipo cadena de caracteres, A$, se utiliza como una subcadena de caracteres:

```
10  REM Escala de diamantes
20  SCREEN 2
30  A$="F40G40H40E40"
40  DRAW "S4BM50,8XA$;"
50  DRAW "S3BM200,8XA$;"
60  DRAW "S2BM50,104XA$;"
70  DRAW "BM200,104XA$;"
100 GOTO 100
```

Fíjate en que el nombre de la variable de tipo cadena de caracteres va seguida de un punto y coma (que nunca deberá omitirse).

Si deseas utilizar una variable numérica dentro de la cadena que sigue a DRAW, para fijar la longitud de la línea, debes anteponer el signo "=" al nombre de la variable, que debe ir seguido por punto y coma. El siguiente programa dibuja quince cuadrados y utiliza la variable X para fijar la escala y el color del cuadrado:

```
10 REM cuadrados crecientes
20 SCREEN 2
30 DRAW"BM31,186"
40 FOR X=1 TO 15
50 DRAW"S=X;C=X;U48R48D48L48"
60 NEXT X
70 GOTO 70
```

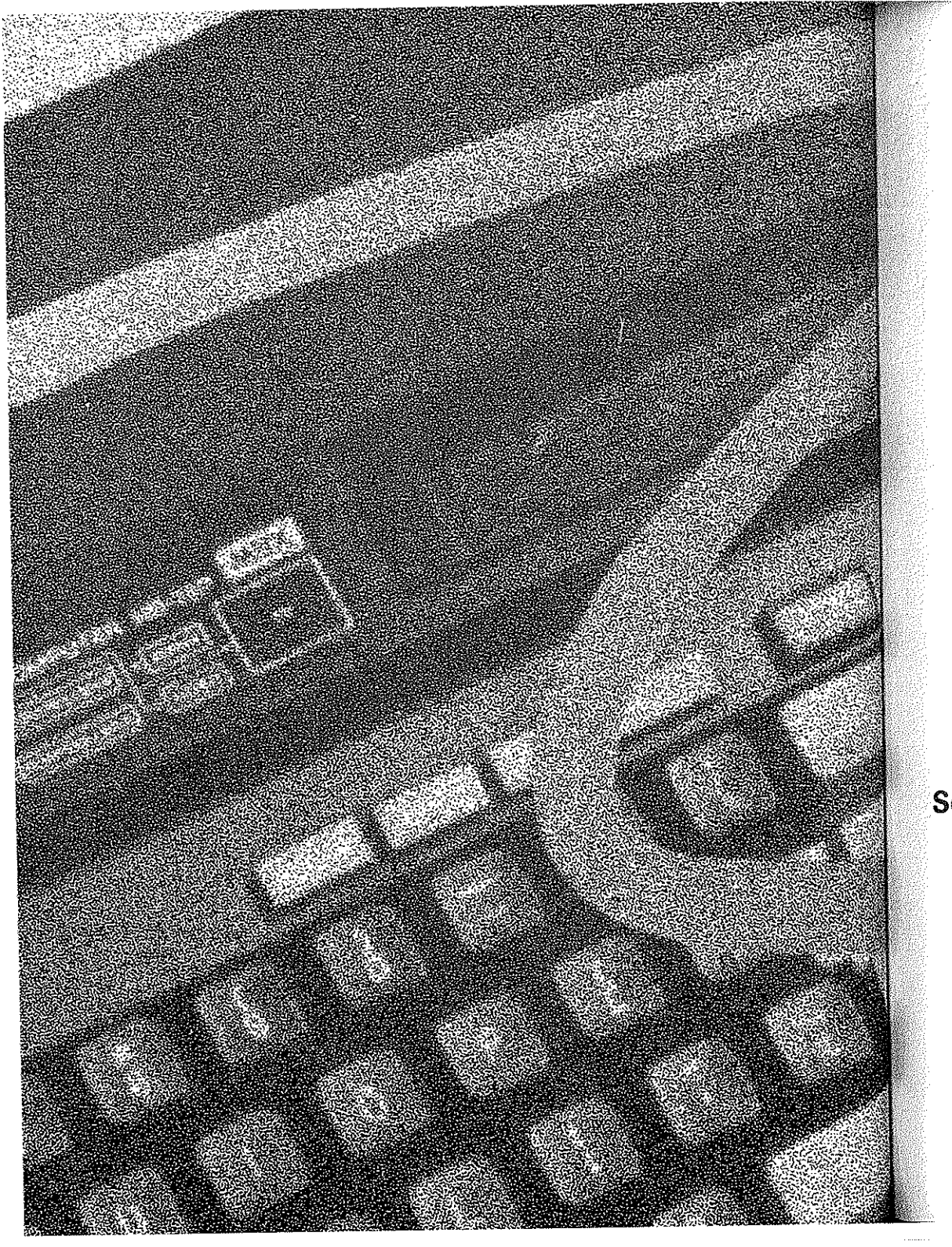
Los lados verticales izquierdos de los cuadrados se han posicionado cuidadosamente en el último punto de un grupo de ocho puntos horizontales. Esta operación se realiza para prevenir la mezcla entre la última línea vertical dibujada en blanco y las catorce líneas horizontales que definen el límite superior de los otros cuadrados. El efecto de mezcla resulta cuando intentas tener más de dos

colores en un grupo de ocho puntos horizontal. (Véase el tema 21 para más detalles.) Para ver el efecto de mezcla cambia la línea 30 por:

```
30 DRAW"BM25,186"
```

Ejercicios

1. Utiliza los comandos S y C para dibujar estrellas en posiciones aleatorias de pantalla; cada estrella, con un color y tamaño distintos.
2. Dibuja una casa.



27
Pinta

El comando PAINT puede utilizarse para rellenar de un determinado color una zona completa delimitada por una línea.

Las coordenadas del comando PAINT deben ser las de un punto que se encuentra en el interior de la figura que quieres pintar. Las coordenadas pueden ser absolutas o relativas.

Los colores de PAINT permitidos son distintos en los dos modos gráficos:

Screen 2

Antes de utilizar PAINT, primero debes dibujar la línea que delimita a la figura, empleando uno de los comandos gráficos, LINE, CIRCLE y DRAW. En esta pantalla el color de PAINT debe ser el mismo color que el especificado en los comandos gráficos. Por ejemplo:

```
10  REM Paralelogramo
20  SCREEN 2
30  PRESET (100,100)
40  LINE -STEP(-50,50),6
50  LINE -STEP(100,0),6
60  LINE -STEP(50,-50),6
70  LINE -STEP(-100,0),6
80  PAINT (110,110),6
90  GOTO 90
```

Las coordenadas (110,110) se encuentran dentro del paralelogramo. Si dibujas la línea de fuera del paralelogramo en rojo, has de pintarlo de rojo. Si intentas pintar el paralelogramo en cualquier otro color, entonces se pintará toda la pantalla de ese color.

Si hubieras pintado la línea exterior en el color del texto, entonces no sería necesario especificar el color en la sentencia PAINT, ya que si no se especifica ningún color se utiliza el color del texto en ese momento. Por ejemplo:

```
10 REM un circulo
20 SCREEN 2
40 COLOR 10
50 CIRCLE(100,100),50
60 PAINT STEP(0,0)
70 GOTO 70
```

La línea 40 fija el color del texto en amarillo. La línea que rodea el círculo se dibujará en amarillo y luego se pinta de amarillo.

Fíjate en que las coordenadas de la instrucción PAINT son el centro del círculo.

En la pantalla 2 has de tener cuidado para evitar los efectos de mezcla, cuando emplees el comando PAINT. El siguiente programa dibuja dos triángulos, uno al lado del otro, para formar un cuadrado. Uno de los triángulos está pintado en amarillo y el otro en negro:

```
10  REM Triangulos V1
20  SCREEN 2
30  COLOR 10
40  PSET (48,50)
50  DRAW "D96R96H96"
60  PAINT (60,70)
70  COLOR 1
80  PSET (70,60)
90  DRAW "R96D96H96"
100 PAINT (70,60)
110 GOTO 90
```

Cuando ejecutes este programa obtendrás un efecto de zigzag; esto es debido a que en la pantalla 2 cada grupo horizontal de ocho puntos no puede tener más de dos colores. No hay ningún problema cuando pintes el triángulo amarillo: el texto es amarillo y el fondo azul (del color prefijado). Cuando dibujes la línea exterior del triángulo negro el color del fondo es todavía azul, pero el nuevo color del texto es negro. Sólo afecta a los grupos de ocho puntos que están en ambos triángulos; el color del texto del grupo completo se cambia a negro. De esto resulta que una parte del triángulo amarillo se vuelve negro, produciendo un efecto de zigzag.

Hay un remedio simple para este problema. Ejecuta el siguiente programa:


```
10  REM triangulos V2
20  SCREEN 2
30  COLOR 10
40  PSET(48,50)
50  DRAW"D96R96U96L96"
60  PAINT(60,70)
70  COLOR 1
80  PSET(48,50)
90  DRAW"R96D96H96"
100 PAINT(70,60)
110 GOTO 110
```

En este programa se dibuja y se pinta un cuadrado en amarillo. Sobre este cuadrado se dibuja y se pinta un triángulo negro. Como el cuadrado sólo contiene un color, no surge ningún problema cuando se dibuja el triángulo negro.

Screen 3

En esta pantalla puedes pintar una figura de cualquier color y pintar alrededor de ella un borde:

```
10 REM pintar con borde
20 SCREEN 3
30 LINE(20,20)-(100,100),7,B
40 PAINT(50,50),9,7
50 GOTO 50
```

Después de la instrucción PAINT se dan dos números. El primero indica el color con el que se va a pintar la figura —puede ser cualquier color—. El segundo número proporciona el color del borde. El color para el borde debe ser el mismo de la línea que rodea la figura. En el ejemplo anterior el cuadrado estaba originalmente pintado en cian y, por tanto, el borde también debe estar en cian. Si intentas utilizar cualquier otro color para el borde, entonces se pintará con él toda la pantalla.

Si la línea que rodea la figura está dibujada en el color del texto, entonces debes especificar el color del borde en ese mismo color, si habías especificado un color distinto para pintar el resto de la figura. Ejecuta el siguiente programa:

```
10  REM Triangulo
20  SCREEN 3
30  COLOR 15
40  LINE (20,20)-STEP(200,100)
50  LINE -STEP(-200,0)
60  LINE -(20,20)
70  PAINT (50,80),10,15
80  GOTO 80
```

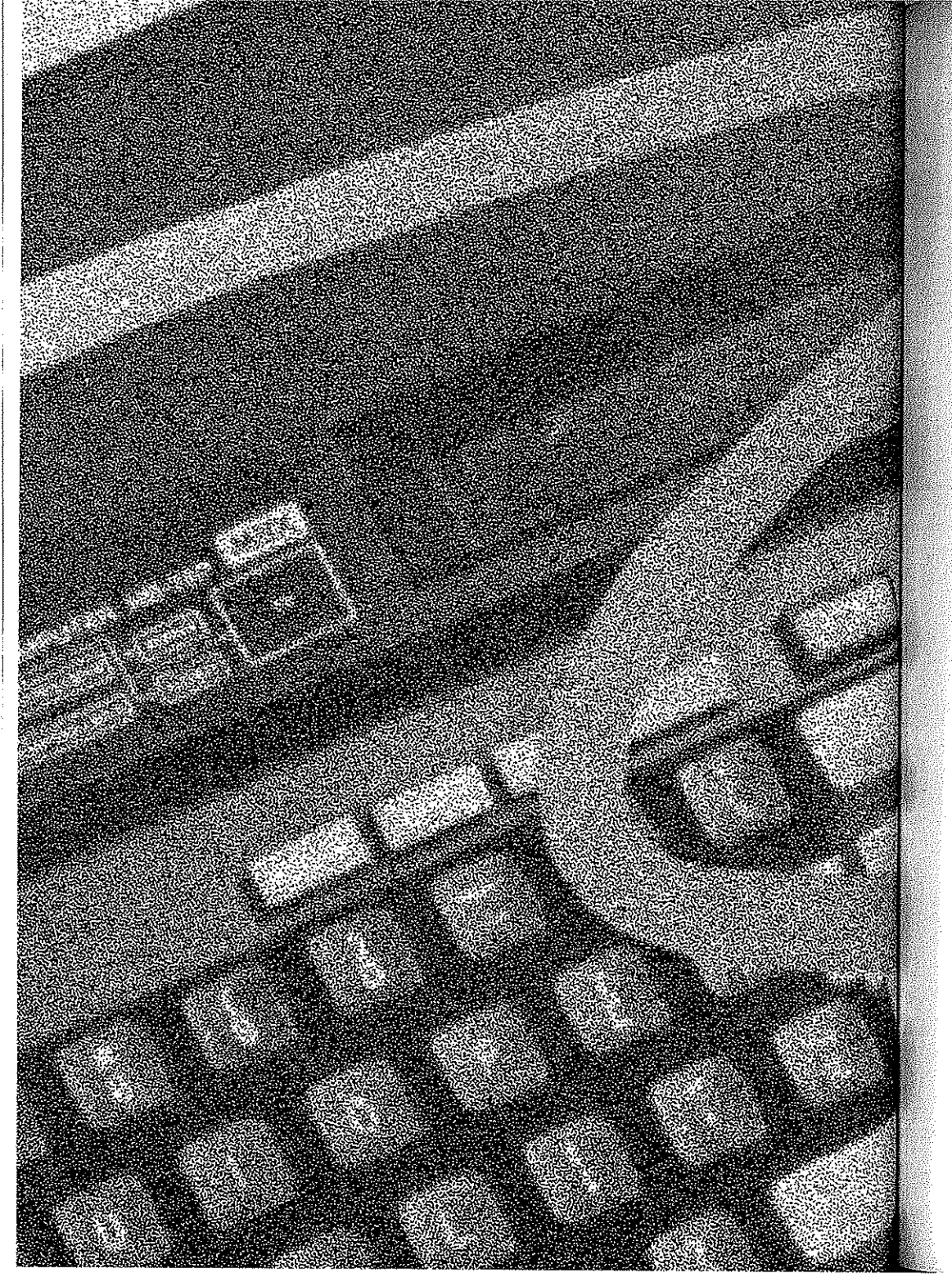
Verás un triángulo en blanco y pintado en amarillo con el borde blanco. Si el borde no se ha fijado en blanco, la orden PAINT cubrirá toda la pantalla.

Si quieres todo el triángulo blanco sustituye la línea 70 por:

```
70 PAINT(50,80)
```

Ejercicios

1. Escribe un programa que dibuje un cuadrado negro en pantalla, utilizando la opción "BF" de la instrucción LINE. Dibuja otro cuadrado negro al lado del anterior, mediante la opción "B" de la instrucción LINE. Ejecuta el programa en las dos pantallas SCREEN 2 y 3.
2. Escribe en la pantalla 2 un programa que dibuje círculos concéntricos; cada uno de un color distinto. No te olvides evitar el efecto borroso; empieza con la instrucción CIRCLE de la circunferencia mayor.



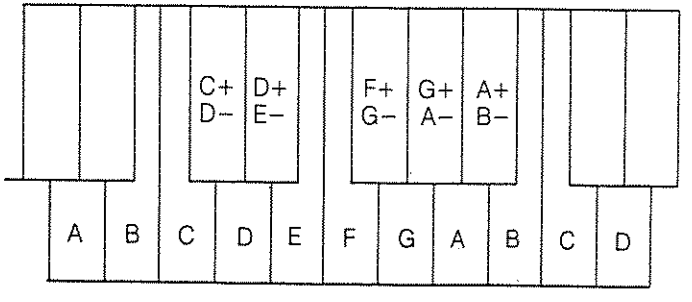
28

El macrolenguaje musical

Si estás familiarizado con la notación musical tradicional, el comando PLAY te permitirá escribir o transcribir música, copiándola de una partitura, de una forma muy sencilla.

El macrolenguaje musical es muy similar al macrolenguaje gráfico (véase el tema 26). La palabra clave PLAY va seguida de una cadena de caracteres que contienen los diversos subcomandos permitidos dentro del macrolenguaje musical.

Los subcomandos A, B, C, D, E, F y G reproducen las notas musicales en este orden: LA, SI, DO, RE, MI, FA y SOL, respectivamente. Cualquiera de estos subcomandos puede ir seguido por los signos "+" o "−", con objeto de indicar si la nota está sostenida o bemolada. La nota, ya sea con sostenido o bemol, debe corresponderse con una nota negra del piano.



Ahora tenemos todas las notas comprendidas en una octava. Para seleccionar la octava deseada debes utilizar el subcomando O. La primera nota en la octava es C(DO) y la última es B(SI). El subcomando de octava debe ir seguido por un número comprendido entre 1 y 8. La octava prefijada inicialmente es 04, donde la primera nota, C, corresponde al DO medio del piano.

Una vez que hayas fijado una octava ésta será la misma para todas las notas siguientes que haya en el programa. El siguiente programa toca la escala en Mi Mayor, entre dos octavas:

```
10 REM Mi Mayor. Notacion A-G
20 PLAY "EF+G+AB05C+D+EF+G+AB06C+D+E"
```

Si ejecutas este programa de nuevo la escala empezará con la sexta octava. Para evitarlo añade el comando 04 al principio de la cadena de caracteres. La línea 20 queda entonces:

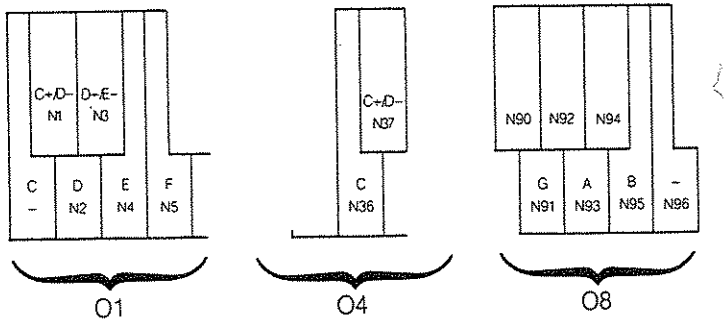
```
20 PLAY"04EF+G+AB05C+D+EF+G+AB06C+D+E"
```

También puedes tocar una nota cualquiera de las ocho octavas mediante el subcomando N. Este debe ir seguido por un número comprendido entre 0 y 96, inclusive. Ya que hay 96 notas en las ocho octavas, la diferencia entre N62 y N63, por ejemplo, es un semitono.

De hecho, las ocho octavas tocadas con el subcomando N varían en un semitono a las tocadas con el comando O. Ya que N0, que se debería corresponder con C en O1, es muda puede utilizarse como una pausa. La menor nota que se puede tocar mediante esta notación corresponde a C+ en O1.

La nota más alta con el comando O es B O8. N96 es el semitono más agudo y toca la C.

Observa que el DO medio (C) es N36.



El siguiente programa toca la misma escala en Mi Mayor, pero esta vez utilizando el comando N:


```
10 REM Mi Mayor. Notacion N
20 PLAY "N41N43N45N46N48N50N52"
30 PLAY "N53N55N57N58N60N62N64N65"
```

Si sabes leer música, encontrarás indudablemente mucho más sencilla la notación A-G.

Hasta ahora todas las notas han tenido la misma duración. Para cambiar la duración de las notas utiliza el subcomando L. Este debe ir seguido por un número que puede tomar cualquier valor entre 1 y 64. Estos son los valores más útiles:

Ln	Duración de la nota
L1	Toca una redonda
L2	Toca una blanca
L4	Toca una negra
L8	Toca una corchea
L16	Toca una semicorchea
L32	Toca una fusa
L64	Toca una semifusa

También puedes emplear L3, L5, etc. L3 producirá un tercio de la nota entera y puede utilizarse para hacer tresillos.

Si no fijas la duración de las notas, éstas se tocarán automáticamente como una negra; o sea, L4 es el valor prefijado de duración de una nota.

Para cambiar la escala en Mi Mayor a corchea, inserta L8 el principio de la cadena de caracteres de la instrucción PLAY.

El comando L afecta a todas las notas siguientes en el programa.

El siguiente programa toca la escala Mi Mayor en corcheas utilizando primero la notación A-G y a continuación la notación N. Observa que la duración de las notas sólo las fijamos una vez en todo el programa:

```
10 REM Mi Mayor. Corcheas
20 PLAY "L8EF+G+AB05C+D+E"
30 PLAY "N41N43N45N46N48N50N52N53"
```

Si sólo quieres fijar la duración de una única nota suprime el subcomando L y pon el número que especifica la duración de la nota a continuación de la nota que va a tocarse.

A+16 es equivalente a L16A+.

Esto no lo podrás hacer con la notación N.

Para indicar una pausa se emplea el subcomando R. El valor prefijado para este comando es R4, que es igual a la pausa de una negra. Para cambiar la duración de la pausa pon un número entre 1 y 64, después del subcomando R. Este número fija la duración de la pausa. Por tanto:

Rn	Duración de la pausa
R1	Pausa de redonda
R2	Pausa de blanca
R4	Pausa de negra
R8	Pausa de corchea
R16	Pausa de semicorchea
R32	Pausa de fusa
R64	Pausa de semifusa

Si deseas tocar una nota o pausa con puntillo añade un punto a continuación del subcomando de la nota. Cada punto después de la nota hace que su duración se incremente la mitad de su duración original. Por ejemplo:

```
10 REM Notas con puntillo
20 PLAY "AR64A.R64A.."
```

toca la nota A como negra, después como una negra con puntillo y, finalmente, como una negra con doble puntillo.

Incluimos las pausas para que puedas escuchar las notas separadas. Sin estas pausas oirías la nota A de forma continua.

Puedes fijar el *tempo* de una pieza musical mediante el subcomando T. El número que sigue a este subcomando debe estar comprendido entre 32 y 255; fija el número de corcheas que se tocan en un minuto. Es decir, el *tempo* de una pieza puede ir desde lo pausado (32 corcheas por minuto) hasta lo vertiginoso (255 corcheas por minuto).

El *tempo* estándar es de T120.

En el último ejemplo de la escala en Mi Mayor cambiamos todas las notas a corcheas y la escala sonó dos veces más deprisa que en el primer ejemplo. Un modo alternativo de acelerar la escala sería fijar el *tempo* en T240. Las notas son, ahora, corcheas, pero cada una se toca con la mitad de la duración anterior.

```
10 REM Mi Mayor. Tempo mas rapido
20 PLAY "T240EF+G+ABO5C+D+E"
30 PLAY "N41N43N45N46N48N50N52N53"
```

El *tempo* sólo tiene que fijarse una vez en todo el programa.

El subcomando V establece el volumen. El número que sigue a este subcomando debe estar comprendido entre 0 y 15, inclusive. V0 es muy bajo o pianísimo; V15 es muy alto o fortísimo.

El valor prefijado para el volumen es V8.

Como ocurría con los otros subcomandos O, L y T, una vez que se ha fijado un valor éste permanece invariable hasta que se le asigne otro o se desconecte el ordenador.

Si hay una parte que se repite a menudo en una pieza puedes asignarla a una variable de tipo cadena de caracteres. Para ejecutar el contenido de variables de este tipo dentro de una instrucción PLAY has de poner el comando X delante del nombre de la variable y un punto y coma a continuación de ella.

Si empleas variables numéricas dentro de la instrucción PLAY éstas han de ir precedidas por el símbolo "=" y seguidas de un punto y coma.

En el siguiente ejemplo empleamos dos variables de tipo cadena de caracteres, A$ y B$. El *tempo* se da en una variable numérica TEMPO; así observarás el efecto que tiene el alterar su valor. Prueba con el valor de TEMPO 120 y 240:

```
10   REM Dame un trago: melodia
20   INPUT "TEMPO";TEMPO
30   A$="V8AR64AR64AB-2R64B-05C04B-AGA"
40   B$="B-05C04FB-A2GF2.F2."
50   GOSUB 1000
60   GOSUB 1000
70   PLAY "V9O5CR64C04A05CF2CR64C04A05CR64C2"
80   PLAY "CD2CR64C04B-AR64A2.G2."
90   GOSUB 1000
100  END
990  REM Subrutina
1000 PLAY "T=TEMPO;XA$;"
1010 PLAY "XB$;"
1020 RETURN
```

Cuando escribas este programa asegúrate de que, para los subcomandos de octava, escribes la letra O. Ten cuidado también con los puntos y comas y los puntos.

Observa cómo hemos fijado individualmente la duración de las notas que son iguales.

Cuando programamos esta melodía decidimos poner cada frase musical en una instrucción PLAY, para facilitar la revisión de errores. Esto trae como consecuencia líneas muy largas; no te olvides de que sólo puedes tener 255 caracteres por línea.

El MSX puede producir tres sonidos simultáneamente. Así que podemos añadir dos voces más a nuestra melodía. El siguiente programa te muestra cómo hacerlo:

```
10   REM Dame un trago: tres voces
20   INPUT "TEMPO";TEMPO
30   A1$="V9AR64AR64AB-2R64B-05C04B-AGA"
40   A2$="V7FEDCDEFEFEF"
50   A3$="V703F2.G2.AGFB-A"
```

```
60   B1$="B-05C04FB-A26F2.F2."
70   B2$="EC2DRC03B-R04CDCDC"
80   B3$="GA02B03C2.F2.C2."
90   GOSUB 1000
100  GOSUB 1000
110  PLAY "V1005CR64C04A05CF2CR64C04A05CR64C2","V8
EF2GF2GF2EGF","V8B-F2ED2EF2GA2"
120  PLAY "CD2CR64C04B-AR64A2.G2.","E-DEFR64F2R64F
R64F2.E2.","GD-2AG02GAB2.03C2."
130  GOSUB 1000
140  END
990  REM Subrutina
1000 PLAY "T=TEMPO;XA1$;","T=TEMPO;XA2$;","T=TEMPO
;XA3$;"
1010 PLAY "XB1$;","XB2$;","XB3$;"
1020 RETURN
```

En cada instrucción PLAY ponemos las cadenas de caracteres asociadas con cada una de las tres voces. Estas deben ir separadas por comas. Observa que el *tempo*, volumen, etc., se fijan independientemente para cada voz.

La voz 1 tiene un volumen mayor, puesto que toca la melodía.

Quizá hayas notado que al final de cada frase las voces se desincronizan. Esto se debe a que las cadenas cada vez contienen diferente número de comandos; algunas tienen más pausas y octavas que otras. Cada comando toma un pequeño pero finito tiempo de ejecución. Puedes intentar sincronizar las voces insertando unos cuantos "R64" y cambiando las octavas en los puntos apropiados. (*Nota:* El comando O4 no hará nada si estamos en dicha octava, pero aumenta el tiempo total de ejecución.)

Todas las notas que has tocado han sonado prácticamente en el mismo tono. Puedes cambiar esta circunstancia asignando una onda a la nota (esta onda controlará la calidad de la nota). En la parte tercera tienes las formas de onda ya definidas.

Para seleccionar una onda emplea el comando S, seguido de un número entre 0 y 15. El período o modulación de la onda puedes fijarlo al comando M. Le puedes asignar cualquier valor comprendido entre 1 y 65.535; el valor estándar es 255. Escribe lo siguiente:

```
10 PLAY "S10L1CDE"
```

Oirás las notas C, D y E según marque la décima onda.

Prueba a cambiar el número de onda. Cuando lo hayas entendido cambia la modulación con el comando M. Por ejemplo:

```
10 PLAY "M1000S10L1CDE"
```

Prueba otras modulaciones con la misma onda, M10, M6000, etc.

Experimenta con otros valores de M y S. El orden en que vayan los comandos S y M no tiene importancia, pero deben ir antes de las notas que se deseen tener.

Finalmente veamos el comando BEEP. Es el más simple para que el ordenador produzca sonido. Escribe:

```
BEEP      <RETURN>
```

Cuando se ejecute un BEEP, el ordenador inicializa todas las variables de sonido, por cuanto todos los macrocomandos musiclaes tendrán unos valores estándar.

PARTE SEGUNDA

GUIA AVANZADA DE PROGRAMACION

29

Edición avanzada de programas

El BASIC del MSX posee una gama completa de modos de pantalla, que te permiten editar un programa en cualquier parte de la misma.

Esta sección te introduce en los detalles técnicos de la edición, así como en las formas avanzadas, usando la tecla de control <CTRL>.

Cómo editar

Cuando realizas un programa necesitas escribir una serie de líneas o incluso borrar secciones enteras. El BASIC del MSX tiene cuatro comandos muy útiles que te facilitarán la tarea de editar: LIST, AUTO, RENUM y DELETE.

LIST

Una vez has introducido en la memoria del ordenador tu programa, posiblemente quieras revisarlo antes de la ejecución. Para poder verlo entero usa el comando LIST. Este comando te listará el programa en un orden claro y lógico; puede listarse usando las diferentes modalidades del MSX.

El comando LIST tiene las siguientes variantes, que puedes utilizar según tus necesidades:

LIST	Lista el programa entero.
LIST <línea>	Lista la línea especificada.

LIST <línea1>—<línea2>	Lista el programa desde la línea 1 hasta la línea 2.
LIST <línea>—	Lista desde la línea especificada hasta el final.
LIST—<línea>	Lista desde el principio hasta la línea especificada.

Te advertimos que si el programa es muy largo el comando LIST seguirá listando el programa en pantalla hasta que llegue la última línea del mismo, o la última línea especificada en el comando.

Hay dos métodos para visualizar la sección deseada del programa:

a) Utilizando LIST <línea1>—<línea2> de manera que sólo obtienes la sección que deseas.
b) Utilizando LIST, y cuando veas la parte buscada pulsas <CTRL> <STOP> y se detendrá el listado.

Notarás que la tecla <STOP> sólo parará temporalmente el listado. Si pulsas la tecla <STOP> otra vez, el listado continuará desde donde terminó.

AUTO

El comando AUTO te introduce en el modo de numeración automática de líneas. Te dará un nuevo número de línea cada vez que se pulse la tecla <RETURN>. Con esto evitas tener que escribir un número para cada línea, facilitando el trabajo del programador. Con este comando los números de línea tienen un incremento constante.

El comando AUTO posee las siguientes variantes que puedes usar según tus necesidades:

AUTO	Numera las líneas comenzando con la línea 10, con incrementos de 10 unidades.
AUTO<línea>	Numera las líneas desde la especificada incrementándose de 10 en 10.
AUTO <línea>,<incremento>	Numera las líneas desde la especificada y con el <incremento> dado.
AUTO,<incremento>	Continuará numerando desde la última línea aparecida con el incremento especificado.
AUTO <línea>,	Numera las líneas desde la especificada con el incremento que tenía antes.

Los dos modos más comunes de usar el comando AUTO cuando escribes o editas un programa son:

1. Cuando comienzas un programa que tienes planificado sobre papel, AUTO se utiliza para numerar líneas desde 10 incrementándose de 10 en 10. Esto da un orden total al programa.
2. Si tienes las líneas 100, 110, 120, 130, 140, 150, 160 y deseas insertar una subrutina entre las líneas 150 y 160. Primero usa RENUM para crear más espacio entre 150 y 160. Esto puedes hacerlo con RENUM 1.000, 150, 100. Como resultado obtendrás las líneas 100, 110, 120, 130, 140, 1000, 1100. Ahora utilizando AUTO 1010, 5 te darán una serie de líneas, comenzando por la 1010 con incremento de 5 sucesivamente. El resultado final será el siguiente:

100, 110, 120, 130, 140, 1000, 1010, 1015, 1020, 1025, ..., 1100

Recuerda estos puntos sobre el comando AUTO:

1. Hay dos modos de salir del modo AUTO:
   a) Pulsar <CTRL><STOP> simultáneamente.
   b) Pulsar <CTRL><C> simultáneamente.

   La línea a partir de la cual te sales no se guarda.
2. Si tecleas un número de línea que ya existía en el programa que estás editando, el ordenador te mostrará una señal de advertencia en forma de asterisco (*) después del número de línea. Si no quieres destruir esa línea pulsa la tecla <RETURN>; pero si lo que deseas es reemplazar dicha línea, entonces ignora la advertencia "*".

RENUM

Una vez editado tu programa encontrarás que los números de línea no se encuentran en forma de incrementos fijos; se ve desordenado, y si el programa se ha de presentar a una segunda persona, seguramente le dará una mala impresión. La solución es bien sencilla: vuelve a numerar el programa entero usando el comando RENUM.

RENUM, por sí solo, vuelve a numerar las líneas a partir de 10 en incrementos de 10. Esta es la forma más común de utilizarlo, pero puedes volver a numerar a partir de un número de línea especificado y con varios incrementos. RENUM cambiará automáticamente los números de línea asociados con GOTO, GOSUB, RESTORE, THEN, ELSE, ON GOTO, ON GOSUB.

El comando RENUM tiene las siguientes variantes que puedes utilizar según tus necesidades:

RENUM	Vuelve a numerar desde la línea 10 en incrementos de 10.
RENUM <línea1>	Vuelve a numerar desde el número de línea especificado por <línea1> en incrementos de 10.
RENUM <línea1>,<línea2>	Renumera con <línea1> la antigua <línea2> y las demás con un incremento de 10.

RENUM <línea1>,<línea2>, <incremento>	Vuelve a numerar con <línea1> la antigua <línea2> continuando con el incremento especificado.
RENUM,<línea2>, <incremento>	Vuelve a numerar a partir del antiguo número de línea (<línea2>), con el nuevo número (10) y el incremento especificado.
RENUM,<línea2>	Vuelve a numerar desde el antiguo número de (línea2), con el número de línea 10, en incrementos de 10.
RENUM,,<incremento>	Vuelve a numerar desde la línea 10, con el incremento especificado.
RENUM<línea1>,, <incremento>	Vuelve a numerar empezando con el nuevo número de línea, a partir de la línea 10, con el incremento especificado.

RENUM también se utiliza para tener una distancia entre línea y línea suficiente, y puedas insertar una sección en el programa, al igual que con AUTO, según vimos anteriormente.

DELETE

A veces encontrarás una sección del programa que quieres eliminar. Si la parte que deseas borrar es sólo una línea, todo lo que tienes que hacer es escribir el número de línea y pulsar <RETURN>. Sin embargo, si deseas borrar varias líneas, utiliza el comando DELETE. Es sencillo y efectivo, pero también es drástico, ya que una vez que has borrado las líneas no hay manera de recuperarlas (excepto si las escribes de nuevo).

El comando DELETE tiene las siguientes variantes que puedes utilizar según tus necesidades:

DELETE <línea>	Borra la línea especificada.
DELETE <línea1>-<línea2>	Borra desde la línea1 hasta la línea2, inclusive.
DELETE-<línea>	Borra todo lo anterior hasta la línea especificada, inclusive.

Teclas de control y de funciones especiales

El MSX tiene un determinado número de teclas de funciones especiales de control que se usan habitualmente en edición. Aquí tienes una tabla que te mostrará qué es lo que hace y cómo se ha de acceder a ellas. A todas las claves de control se tiene acceso pulsando <CTRL> y la tecla correspondiente al mismo tiempo.

Código hexadecimal	Tecla control	Tecla especial	Función
02 *	<CTRL><B>		Mueve el cursor una palabra a la izquierda.
03 *	<CTRL><C>		Detiene la ejecución cuando el BASIC está en espera de datos (INPUT). Vuelve al modo de comando.
05 *	<CTRL><E>		Borra todo lo que hay a la derecha del cursor incluido el carácter sobre el que se encuentra el cursor.
06 *	<CTRL><F>		Mueve el cursor una palabra a la derecha.
07 *	<CTRL><G>		Sonido *(Beep)*.
08	<CTRL><H>	BS	Retrocede y borra un carácter a la izquierda del cursor, desplaza el lado derecho de la misma línea una posición a la izquierda.
09	<CTRL><I>	TAB	Desplaza el cursor hasta la siguiente posición TAB. TAB está fijado en intervalos de 8 caracteres. Deja espacios en blanco desde el lugar donde se ha movido.
0A *	<CTRL><J>		Salto de línea. Mueve el cursor hasta el principio de la línea siguiente.
0B *	<CTRL><K>	HOME	Se desplaza el cursor a la esquina superior izquierda de la pantalla.
0C *	<CTRL><L>	CLS	Limpia la pantalla y mueve el cursor a la posición HOME.
0D *	<CTRL><M>	RETURN	Fin de línea. Introduce la línea actual.
0E *	<CTRL><N>		Desplaza el cursor al final de la línea actual.
12 *	<CTRL><R>	INS	Da entrada al modo INSERCION. El cursor se hace una línea blanca y permite insertar caracteres en la posición de cursor sin borrar lo anterior.
15 *	<CTRL><U>		Borra entera la línea en curso.
18 *	<CTRL><X>	SELECT	Ignorada por esta versión MSX.

* Indica que desactivará el modo inserción si está ON.

Código hexadecimal	Tecla control	Tecla especial	Función
1B *	<CTRL><[>	ESC	Ignorada por esta versión MSX.
1C *	<CTRL><Y>	→	Mueve el cursor un espacio a la derecha.
1D *	<CTRL><]>	←	Mueve el cursor un espacio a la izquierda.
1E *	<CTRL><^>	↑	Mueve el cursor un espacio hacia arriba.
1F *	<CTRL><->	↓	Mueve el cursor un espacio hacia abajo.
7F	<CTRL><DEL>	DEL	Borra el carácter donde está situado el cursor y desplaza la parte derecha de la línea un lugar a la izquierda.

* Indica que desactivará el modo inserción si está ON.

30

Constantes y variables

Constantes

Las constantes son números o cadenas de caracteres que no cambian de valor; por ejemplo, 100 y "HOLA" son constantes.

Hay dos tipos de constantes: numéricas y de cadenas de caracteres.

Constantes de cadena de caracteres

Una constante de cadena de caracteres es una secuencia de 255 caracteres de longitud máxima. Pueden ser caracteres, gráficos o códigos de control utilizados en el BASIC del MSX. Deben estar encerrados por dobles comillas; por ejemplo:

"ORDENADOR MSX"
"¿...DONDE ESTA EL?"
"LUIS Y MANOLO"
"$ 100,000,000"

Constantes numéricas

Son números positivos o negativos. Hay seis tipos:

1. Constantes enteras — Todos los números comprendidos entre −32768 y 32767 ó 111111111111111 y

0000000000000000 en forma binaria. Estas constantes no contienen punto decimal.

2. Constantes con coma — Números reales positivos o negativos que pueden contener un punto decimal; por ejemplo, 657.188.

3. Constantes con coma flotante — Números reales positivos o negativos en forma exponencial.
Formatos:
Precisión simple
<entero>E<entero> −56E10
<comafija>E<entero> 7.865E5
Precisión doble
<entero>D<entero>
−1896243232662D50
<comafija>D<entero>
7.8827726266265D-5
El rango exacto está entre 1E-64 y 1E64.

4. Constantes hexadecimales — Los números hexadecimales se señalan con el prefijo &H. Es el sistema de numeración en base 16; o sea, utiliza los dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.
Ejemplos: &HFF (1 byte de longitud).
&HBD1A (2 bytes de longitud).

5. Constantes en octal — Los números en octal se representan con el prefijo &O. El octal es el sistema de numeración en base 8, que trabaja con los dígitos 0, 1, 2, 3, 4, 5, 6, 7.
Ejemplo: &O6543

6. Constantes binarias — Los números binarios se representan con el prefijo &B. El binario es el sistema de numeración en base 2 emplea los dígitos 0 y 1.
Ejemplo: &B01010101

Precisión simple y doble precisión

Una de las características más significativas del BASIC del MSX es que tienen catorce dígitos exactos en doble precisión para trabajar con funciones aritméticas. La mayoría de los ordenadores con palabras de 8 bits no la tienen. La precisión doble se ofrece en sistema de 16 y 32 bits, pero Microsoft ha reescrito el 4.5 MS-BASIC para darle mayor exactitud de cálculo.

Puedes tener constantes numéricas, tanto de simple como de doble precisión, pero si no especificas cuál, el ordenador asume doble precisión.

Exactitud:	precisión simple	6 dígitos.
	doble precisión	14 dígitos.

Las constantes de precisión simple tienen las siguientes características:

1) Exponencial en forma E — 1.65E-2
2) Con marca de exclamación — 235.77!

Las constantes de doble precisión tienen las siguientes características:

1) Cualquier dígito o número sin exclamación o exponente — 1878932.2 156
2) Exponencial usando D — 652.76533D−06
3) Añadiendo el signo # — 387.001#

Variables

Las variables son nombres a las cuales se les asigna un valor. Estos valores pueden cambiar. Han de ser especificados por el programador (por ejemplo, PI = 3,141596802) o calculados por el ordenador (por ejemplo, PI = 4*ATN(1)).

Nombres de las variables

Los nombres de las variables pueden ser de cualquier longitud. Sin embargo, el ordenador sólo reconocerá los dos primeros caracteres del nombre. Si dos variables tienen los dos primeros caracteres del nombre iguales, se considerará que es la misma; por ejemplo, VAR1 y VAR2 son la misma variable. Si queremos que sean diferentes deberemos nombrarlas como V1 y V2.

Los nombres de variables no deben contener palabras reservadas, o sea, palabras claves del BASIC, en ninguna parte del nombre. Por ejemplo, LENGUA% es incorrecto, ya que tiene la palabra reservada LEN.

Si un nombre de variable comienza por FN, entonces se asume como una función definida por el usuario.

Declaración de tipos

Si a una variable no se le asigna un carácter de declaración de tipo, entonces el ordenador la asume como variable numérica de precisión doble.

A1 = <número de doble precisión>

Sufijos de declaración de tipo:

$ El signo dólar indica que la variable es una cadena de caracteres.
Z$ = "Cadena de caracteres".
DIRECCION$ = "CALLE ALCALA"

% El signo tanto por ciento indica una variable entera
X% = 25
ZXY% = −32768

! Una exclamación indica una variable numérica de simple precisión
F! = 3679.7!
NUM! = 4.2888E−02

El signo almohadilla indica una variable numérica de doble precisión.
H# = 3.141596802134
MAXINUM# = 5277.76525D25.

DEF <tipo de carácter>

Es posible definir un tipo mediante un carácter, usando las diferentes declaraciones DEF. Cuando el tipo de una variable se declara con una de estas funciones, cualquier variable que comience con dicho carácter se convertirá en el tipo del carácter especificado, sin tener que ser declarada así en todos los nombres de la variable. Esto significa que si ponemos DEFSTR A, cualquier variable que comience por el carácter A será una variable de tipo cadena de caracteres, sin tener que usar el signo dólar. Sin embargo, esto no significa que AB% sea una variable de tipo cadena de caracteres, ya que la declaración de variable entera (o sea, %) tiene preferencia sobre DEFSTR.

Hay cuatro declaraciones DEF <tipo de carácter>. Estas son las siguientes:

- DEFSTR declara como una variable de tipo cadena de caracteres a cualquier nombre de variable que comience con los caracteres especificados.

```
DEFSTR A
AST = "PLANETA"
```

- DEFINT declara como entero a cualquier nombre de variable que comience con los caracteres especificados.
- DEFSNG declara como un entero de simple precisión a cualquier nombre de variable que comience con los caracteres especificados.
- DEFDBL declara como un número de doble precisión a cualquier nombre de variable que empiece con los caracteres especificados.

No olvides que las declaraciones con los caracteres #, $, % y ! tienen prioridad absoluta sobre las anteriores.

MATRIZ (DIM)

Una matriz es un grupo de variables que poseen un nombre en común. Se crea utilizando la declaración DIM. Esta reserva un espacio en memoria para cualquier matriz antes de utilizarse. Cada elemento de la matriz tiene un índice o número de etiqueta. Si defines DIM C(3), entonces tienes cuatro variables con el nombre C, que son C(0), C(1), C(2) y C(3). Todas las variables de la matriz comienzan por el elemento de índice cero. Estas son matrices de una dimensión, pero no hay ninguna razón para que no pueda existir una matriz multidimensional, como, por ejemplo, DIM A(2,2), de la que se obtiene una tabla de variables de dimensión 3 × 3, tal como se muestra a continuación:

A(0,0)	A(1,0)	A(2,0)
A(0,1)	A(1,1)	A(2,1)
A(0,2)	A(1,2)	A(2,2)

Esta matriz es de dos dimensiones, pero puedes tener una de hasta diez dimensiones. La dimensión máxima de una matriz es 255, pero puedes desbordar la memoria si intentas crear una tan inmensa.

Si una matriz tiene menos de doce elementos, DIM A(10), no es necesario utilizar la declaración DIM, ya que el ordenador automáticamente reserva una posición de memoria para once elementos. Los programas como el de abajo funcionan correctamente sin utilizar la declaración DIM:

```
10 FOR I = 0 TO 10
20 S(I) = I
30 NEXT I
```

Es posible tener matrices de cualquier tipo, tan largos como el tipo que se declara. Por ejemplo, DIM S$ (100) te dará 101 elementos para una matriz de tipo cadena de caracteres.

Cuando se inicializa una matriz, todos los valores se asumen como cero para matrices numéricas, y cadenas nulas para matrices de tipo cadena de caracteres.

Demanda de memoria

Las variables se almacenan en DIMAREA, VARIABLEAREA o STRINGAREA, dependiendo de su tipo (véase el mapa de memoria).

Aquí tienes una lista del número de bytes ocupados por estas variables:

Variables	Entero	2 bytes
	Precisión simple	4 bytes.
	Doble precisión	8 bytes.
	Cadena de caracteres	3 longitudes máximas de caracteres.
Matrices	Entero	2 bytes por elemento.
	Precisión simple	4 bytes por elemento.
	Doble precisión	8 bytes por elemento.
	Cadena de caracteres	3 longitudes máximas de caracteres por elemento.

VARPTR

Para encontrar exactamente dónde se almacenan los datos de las variables en memoria puedes usar la función especial VARPTR, que te dará la dirección de una variable en particular; por ejemplo:

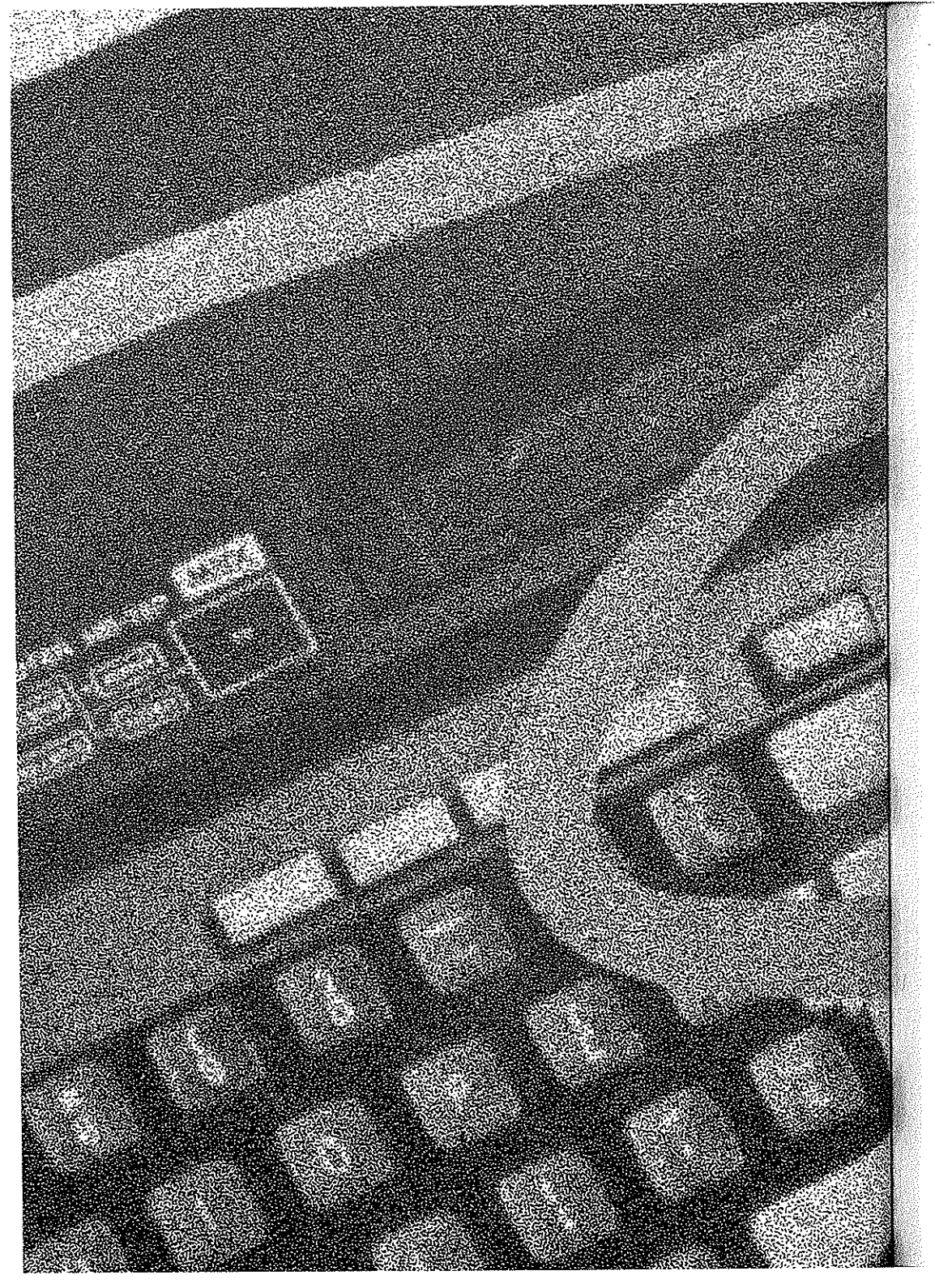
```
10 A% = 100
20 PRINT VARPTR (A%)
```

Puedes encontrar la localización en memoria de cualquier tipo de variables sin importar su longitud. Puede ser una variable de tipo cadena de caracteres o incluso una matriz.

Area de la variable de tipo cadena de caracteres

Todos los elementos de las variables de tipo cadena de caracteres están almacenados en la sección STRINGAREA de la memoria del ordenador (véase el mapa de la memoria). El tamaño de la cadena está restringido a 200 bytes de modo estándar; con ello, la longitud máxima de una variable de este tipo estará limitada a 200 caracteres. Puedes incrementar el tamaño de la STRINGAREA, utilizando la instrucción CLEAR. Por ejemplo, para incrementar la STRINGAREA a 255 bytes por cadena, ejecuta:

```
CLEAR 255
```



31
Conversión de tipos

El BASIC del MSX puede convertir un tipo de constante numérica a otro. Para dicho cambio hay establecidas ciertas reglas:

1. Si una constante numérica de un cierto tipo se iguala a otro, el número almacenado dependerá enteramente del tipo de la variable a que ha sido igualada.

Ejemplo 1:

```
10 C!=1.2345678901
20 PRINT C!
run
 1.23457
```

El número de doble precisión 1.2345678901 se convierte a precisión simple cuando le asignamos a una variable numérica de precisión simple.

Ejemplo 2:

```
10 I%=1.23E-03
20 PRINT I%
run
 0
```

El número de precisión simple 1.23E−03 se redondea al entero más próximo, que es 0, cuando se asigne a una variable entera.

2. Durante la evaluación de una expresión todas las variables y constantes se utilizan con la máxima precisión. Cuando la expresión se dispone a igualar una variable, el resultado de la expresión se convierte al tipo especificado

por la variable. Si no se especifica el tipo de la variable el resultado vuelve al mismo nivel que poseía cuando se evaluaba la expresión.

Ejemplo 1:

```
10 D=1!/3#
20 PRINT D
run
 .33333333333333
```

1! es de precisión simple mientras que 3 # es de doble precisión. La operación aritmética se ejecuta en doble precisión y asigna el resultado a una variable de doble precisión.

Ejemplo 2:

```
10 D=1!/3!
20 PRINT D
run
 .333333
```

Las dos constantes están en precisión simple; por tanto, la operación aritmética se lleva a cabo en dicho formato, pero D es una variable de doble precisión. Por consiguiente, el resultado se convierte a doble precisión, pero no con la misma exactitud.

Ejemplo 3:

```
10 D!=1/3
20 PRINT D!
run
 .333333
```

La operación aritmética se lleva a cabo en doble precisión, pero el resultado se convierte a precisión simple por truncamiento cuando se asigna a D!

3. Los operadores lógicos convierten sus constantes y variables a números enteros de 16 bits antes de que la operación se lleve a cabo. El resultado es un entero y los operandos deben pertenecer al rango de los enteros; en caso contrario se producirá un error de desbordamiento *(Overflow error)*.

Ejemplo:

```
10 D%=156.65 AND 654
20 PRINT D%
run
 140
```

Los números se transforman a 156 y 654. El resultado 140 se asigna a D%, que es una variable entera.

4. Cuando un valor en coma fija o coma flotante se convierte en un entero, la parte decimal se trunca.

Ejemplo 1:

```
10 A%=123.456
20 PRINT A%
run
 123
```

El valor 123.456, que está en coma fija, se trunca asignando como resultado 123 a la variable entera A%.

Ejemplo 2:

```
10 A%=1.0097554D02
20 PRINT A%
run
 100
```

El número en coma flotante 1.009755D02 se redondea a 100 cuando se asigna a la variable A%.

5. Cuando un número de precisión simple se convierte en un número de doble precisión, sólo se pueden integrar al resultado de doble precisión los seis dígitos que tenía en precisión simple.

Ejemplo:

```
10 A#=1.33333
20 PRINT A#
run
 1.33333
```

A# es una variable de doble precisión, pero no hace que al asignarle un número éste sea más preciso. El resultado es también 1.33333.

Conversión de tipos mediante funciones del BASIC del MSX

CDBL, CINT, CSNG, VAR, STR$

Aunque el MSX efectúa conversiones de tipo automáticamente al evaluar expresiones, puedes programarlo para que haga las conversiones, usando alguna de estas funciones. CDBL, CINT y CSNG siguen las reglas anteriores, pero VAR y STR$ son conversiones de cadenas de caracteres a números y de números a cadena de caracteres, respectivamente, y siguen reglas diferentes.

CINT

La función CINT convierte números de precisión simple, de doble precisión, variables y expresiones a números enteros. El argumento debe estar comprendido dentro del rango de los enteros (−32768, 32767). En caso contrario, el resultado producirá un error de desbordamiento *(Overflow error)*.

CINT(<num-const>)
CINT(<num-var>)
CINT(<num-exp>)

Ejemplo:

```
PRINT CINT(1234.56789)
1234
```

CSNG

La función CSNG convierte su argumento en un número de precisión simple.

CSNG redondea a seis dígitos decimales. Si es un entero la precisión seguirá siendo la misma.

```
CSNG(<num-const>)
CSNG(<num-var>)
CSNG(<num-exp>)
```

Ejemplo:

```
PRINT COS(0.7656)
.72096670541357
PRINT CSNG(COS(0.7656))
.720967
```

CDBL

La función CDBL transforma números enteros, de precisión simple, variables y expresiones, a números de doble precisión. Cuando se pasa un número de precisión simple a doble, sólo se pueden integrar al resultado seis dígitos.

```
CDBL(<num-const>)
CDBL(<num-var>)
CDBL(<num-exp>)
```

Ejemplo:

```
PRINT CDBL (5.666!)
5.666
```

VAL Y STR$

Estas funciones se utilizan para convertir cadenas de caracteres a números y viceversa.

STR$(<número>)

STR$ convierte un argumento numérico en una cadena de caracteres.

VAL(<cadena de caracteres>)

VAL devuelve el valor numérico de un carácter. Si la cadena de caracteres es la representación de un número, entonces se puede utilizar VAL. Sin embargo, no puede operar una expresión dentro de una cadena. VAL puede eliminar espacios, tabulaciones y saltos de línea delante de un número en una cadena, pero no los puede sustituir por otros caracteres. VAL reconoce los signos más y menos.

Véanse los ejemplos del tema 15 en la parte primera.

32

Operadores y expresiones

Una expresión puede ser una cadena de caracteres, una constante numérica, una variable o cualquier combinación de ellas que devuelve un único valor, y tan larga como esté permitido. Hay cuatro tipos de operadores matemáticos y lógicos:

1. Aritméticos.
2. De relación.
3. Booleanos (lógicos).
4. Funciones.

El **1** y el **2** se explican en este tema. (Véanse las partes pertinentes para los otros dos apartados.)

Operadores aritméticos

Los operadores aritméticos, en orden de preferencia, son:

1.	^	exponenciación	A ^ B
2.	–	negación	– A
3.	*,/	multiplicación y división	A*B, A/B
4.	+,–	adición y sustracción	A + B, A – B

Para cambiar el orden de preferencia en el cálculo emplea los paréntesis ( ). Dentro de los paréntesis se mantendrá el orden de preferencia.

DIV y MOD (división entera)

DIV se denota en el MSX por el signo ¥; realiza la división entera. MOD es el resto de dicha división entera.

¿Qué son DIV y MOD? Aquí tienes un ejemplo ilustrativo. En una división normal si divides 10 entre 4 obtienes 2,5.

Igualmente,

13/5=2,6

En la división entera, sin sacar decimal, se obtiene un 2 en el cociente y un 3 en el resto.

En el BASIC del MSX, si haces 13 DIV 5 obtendrás:

13¥5=2 ... parte entera de la división.

Si haces 13 MOD 5=3 ... resto de la división.

El cociente y el resto se obtienen con 16 bits. Antes de que la operación tenga lugar, cualquier operando no entero se pasará a entero mediante truncamiento; esto es, todas las partes decimales desaparecen.

16.78¥5.89=3 (lo mismo que 16¥5=3)
16.78 MOD 5.89=1 (lo mismo que 16 MOD 5=1)

MOD sigue a DIV(¥) en orden de preferencia.

Ejemplo:
Aquí te mostramos el orden de preferencia de MOD y DIV.

```
   158 MOD 789 DIV 10
1.         789 DIV 10
              78
2. 158 MOD    78
       20
```

Es decir, 1580 MOD 789¥10=20

Operadores de relación

Los operadores de relación comparan dos valores y devuelven una respuesta verdadera (−1) o falsa (0). Se emplean normalmente en las condiciones IF THEN, pero pueden incorporarse en una expresión.

OPERADOR	RELACION	EJEMPLO
=	Igual a	A=B
<>	Distinto a	A<>B
<	Menor que	A<B
>	Mayor que	A>B
<=	Menor o igual que	A<=B
>=	Mayor o igual que	A>=B

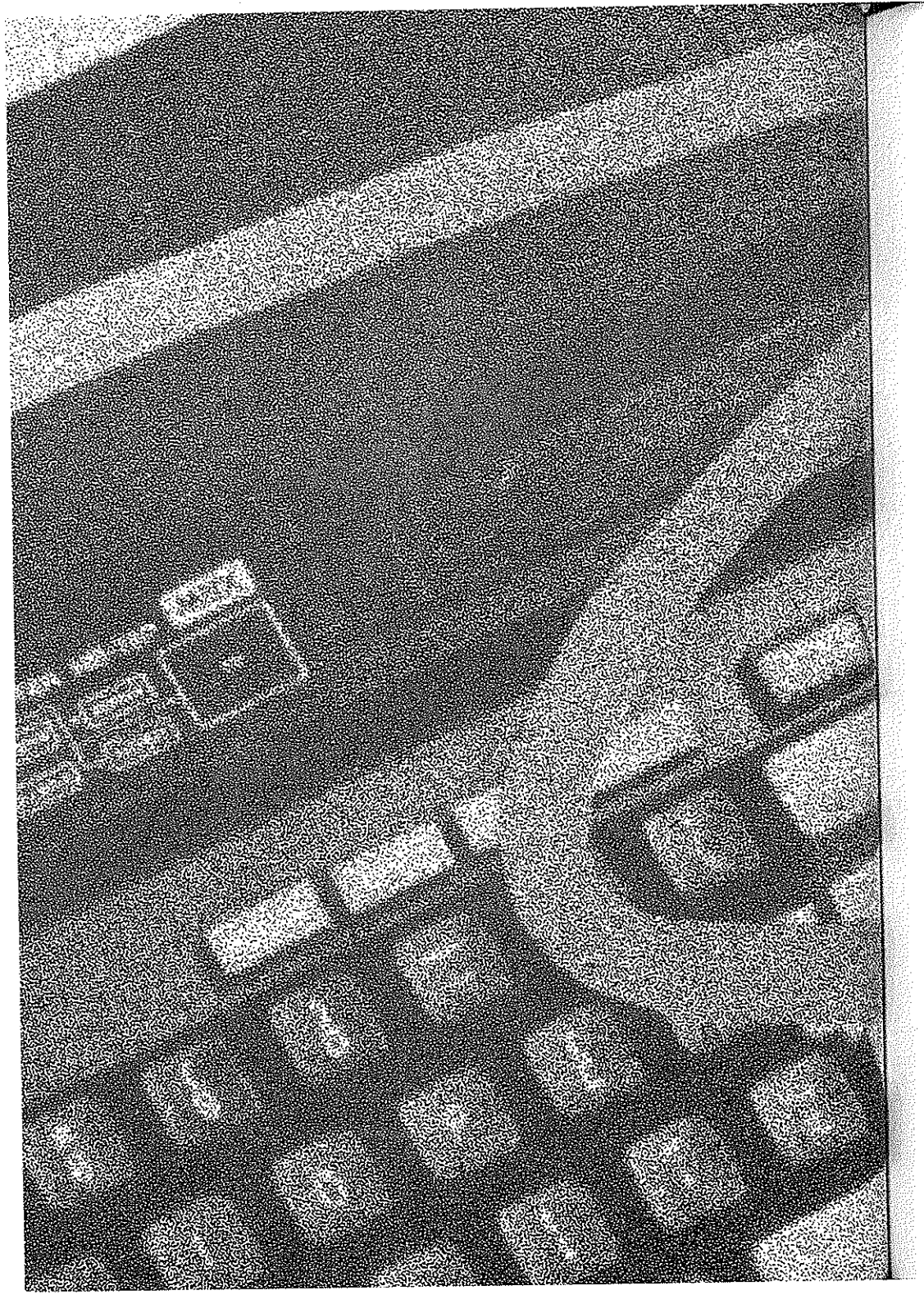
Verdadera devuelve el valor −1. Falsa devuelve el valor 0.

Si combinas operadores de relación y aritméticos, estos últimos tendrán preferencia.

Ejemplo:

A*B=C.
Primero calcula A*B, y el resultado se compara con C.

Los operadores de relación se explican extensamente en la sección IF/THEN/ELSE (parte segunda, tema 35).



33

Presentación de los sistemas de numeración del MSX

Como la mayoría de los humanos que vivimos en este planeta tenemos diez dedos en las manos, utilizamos el sistema numérico decimal. A diferencia de los *homo-sapiens*, los ordenadores, que son básicamente una inmensa matriz de interruptores, pueden tener sólo dos "dedos", ON u OFF (en otras palabras, 0 ó 1). Por tanto, los ordenadores trabajan en binario internamente. Para beneficio del usuario, el BASIC del MSX dispone de números decimales, pero naturalmente también puede trabajar con números binarios en los programas de los usuarios. Aparte del sistema binario el MSX puede utilizar el octal y el hexadecimal. Todos los números hexadecimales, binarios y octales son enteros, y deben estar dentro del rango de los enteros; es decir, entre −32768 y 32767.

Las funciones aritméticas se calculan en el sistema decimal codificado en binario (BCD), que explicaremos más adelante.

Binario

Definición: Sistema numérico en base 2. Utiliza el 0 y el 1 para representar todos los números.

BASIC del MSX: &B <binario> representa un número binario.
BIN$ pasa de decimal a binario.

A cada dígito de un número binario se le llama "bit". A 8 bits binarios se les denomina "byte". Un byte es el tamaño de una dirección de memoria en el sistema MSX. Un byte binario puede tener cualquier valor entre 00000000 y 11111111; o sea, desde 0 a 255 en decimal.

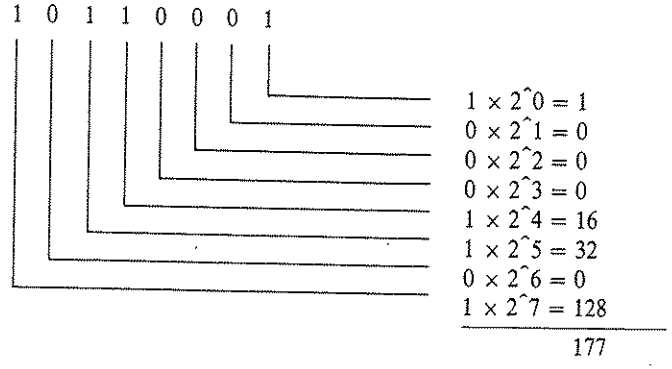
DECIMAL	BINARIO	CON 8 BITS
0	0	00000000
1	1	00000001
2	10	00000010
3	11	00000011
4	100	00000100
5	101	00000101
6	110	00000110
7	111	00000111
8	1000	00001000
9	1001	00001001
10	1010	00001010
11	1011	00001011
12	1100	00001100
13	1101	00001101
14	1110	00001110
15	1111	00001111
16	10000	00010000

Conversión de decimal a binario y de binario a decimal

Cómo convertir números de 8 bits binarios al equivalente decimal.

NUMERO DE BIT	CORRESPONDE A	VALOR
7	2^7	128
6	2^6	64
5	2^5	32
4	2^4	16
3	2^3	8
2	2^2	4
1	2^1	2
0	2^0	1

Ejemplo: Convertir 10110001 a decimal.



Binario decimal
&B1011001 = 177

Cómo convertir un número decimal a binario.

Ejemplo: convertir 53 a binario.

BIT	NUMERO			
7	2^7	128		0
6	2^6	64		0
5	2^5	32	53 − 32 = 21	1
4	2^4	16	21 − 16 = 5	1
3	2^3	8		0
2	2^2	4	5 − 4 = 1	1
1	2^1	2		0
0	2^0	1	1 − 1 = 0	1

Agrupando todos los bits binarios obtienes 53 = &B00110101.

En BASIC:

Cómo convertir números de 8 bits a su equivalente decimal.

Ejemplo:

```
10 X%=&B00101011
20 PRINT X%
run
 43
```

Cómo convertir un número decimal a su equivalente binario:

Ejemplo:

```
10 X%=171
20 PRINT BIN$(X%)
run
10101011
```

Octal

Definición: Sistema de numeración en base 8. Utiliza los dígitos entre el 0 y el 7 para representar todos los números.

BASIC del MSX: &O<octal> representa un número octal.
OCT$ pasa de decimal a octal.

DECIMAL	OCTAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20

Conversión de octal a decimal y de decimal a octal

Cómo convertir un número octal a su equivalente decimal.

<decimal> = <1.er dígito> *8^0 + <2.º dígito> *8^1 + ...
... <4.º dígito> *8^3 + ...

Ejemplo: Octal &O5436 a decimal

6*8^0 + 3*8^1 + 4*8^2 + 5*8^3 = 2846
&O5436 = 2846

Cómo convertir un número decimal a octal.

Ejemplo: Decimal 6588 a octal.

DIGITO	DECIMAL	DIVISIÓN ENTERA	MODULO OCTAL
4	8^4 4096	6588 DIV 4096 =	1 6588 MOD 4096 = 2492
3	8^3 512	2492 DIV 512 =	4 2492 MOD 512 = 444
2	8^2 64	444 DIV 64 =	6 444 MOD 64 = 60
1	8^1 8	60 DIV 8 =	7 60 MOD 8 = 4
0	8^0 8	4 DIV 1 =	4

Por tanto obtienes &O14674 = 6588 (decimal).

En el BASIC del MSX

Cómo convertir un número octal en su equivalente decimal.

Ejemplo:

```
10 X%=&O6517
20 PRINT X%
run
 3407
```

Cómo convertir un número decimal en su equivalente octal.

Ejemplo:

```
10 X%=171
20 PRINT OCT$(X%)
run
253
```

Hexadecimal

Definición: Sistema de numeración en base 16. Utiliza los dígitos entre 0 y 9, y adicionalmente A, B, C, D, E y F para representar todos los números hexadecimales.

BASIC del MSX: &H <hexadecimal> representa un número hexadecimal.
HEX$ de decimal a hexadecimal.

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

Conversión de hexadecimal a decimal y de decimal a hexadecimal

Cómo convertir un número hexadecimal en su equivalente decimal.

<decimal> = <1.er dígito>*16^0 + <2.º dígito>*16^1
+ <3.er dígito>*16^2 + <4.º dígito>*16^3 ...

Ejemplo: Pasar &HFF a decimal.

(&HF)*16^0 + (&HF)*16^1 = (15)*1 + (15)*16 = 255
&HFF = 255

Cómo convertir un número decimal en su equivalente hexadecimal.

Ejemplo: Pasar 7752 a hexadecimal.

DIGITO DECIMAL			DIVISION ENTERA	HEXADECIMAL	MODULO
3	16^3	4096	7752 DIV 4096 =	1	7752 MOD 4096 = 3656
2	16^2	256	3656 DIV 256 =	E(14)	3656 MOD 256 = 72
1	16^1	16	72 DIV 16 =	4	72 MOD 16 = 8
0	16^0	1	8 DIV 1 =	8	

Por tanto obtienes &H1E48 = 7752 (decimal).

En BASIC

Cómo convertir un número hexadecimal a su equivalente decimal.

Ejemplo:

```
10 X%=&HEFA3
20 PRINT X%
run
-4189
```

Cómo convertir un número decimal a su equivalente hexadecimal.

Ejemplo:

```
10 X%=171
20 PRINT HEX$(X%)
run
AB
```

Notas:

1. El sistema binario se emplea, sobre todo, para diseñar *sprites*, ya que es más fácil ver los bits que indican si ese punto está visto u oculto.
2. El hexadecimal se usa para las posiciones de memoria y el código máquina.

Resumen de decimal, binario, octal y hexadecimal

DECIMAL	BINARIO	OCTAL	HEXADECIMAL
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Sistema decimal codificado en binario (BCD)

El MSX dispone de catorce dígitos exactos en precisión doble BCD para funciones matemáticas. Esto hace que las operaciones matemáticas no demasiado largas generen errores de redondeo invariablemente asociados a muchos sistemas de 8 bits. Todas las operaciones aritméticas y funciones se calculan con esta exactitud a menos que en el programa del usuario se especifique lo contrario.

¿Por qué se usa el decimal codificado en binario? ¿Cuál es su ventaja y por qué es más exacto? Para empezar, vamos a ver cómo se trabaja con los números en el sistema binario, que es el más conveniente para los ordenadores.

En binario, si intentas expresar un número menor que 1 (o sea, uno con un punto decimal) a veces obtendrás errores. Hay algunos números que se pueden expresar en decimal, pero no en binario.

Si queremos expresar 0,625 en binario:

$$0{,}625 = 1/4 + 1/8 = 1(2\hat{}2)\ 1/(2\hat{}3)$$
$$0{,}625 = 2\hat{}(-2) + 2\hat{}(-3)$$

Por tanto, puedes convertir 0,625 a binario sin ningún problema. Sin embargo, si tienes un número como 0,1, no puedes hacer esto. No hay forma de expresar 0,1 con exactitud en binario. Esto te mostrará el porqué:

$$0{,}1 \sim 1/16 + 1/32 + 1/256 + 1/512 + 1/4096 + 1/8912 \ldots \sim 0.0999 \ldots$$

ó

$$0{,}1 \sim 1/(2\hat{}4) + 1/(2\hat{}5) + 1/(2\hat{}6) + 1/(2\hat{}7) \ldots \sim 0.09999 \ldots$$

No puedes obtener 0,1 en binario; en su lugar obtienes una serie.

En vez de pasar un decimal completamente a binario, el MSX utiliza el decimal codificado en binario.

En el sistema codificado en binario un dígito decimal se representa con 4 bits binarios. Esto significa que 0001 corresponde al 1, 0010 corresponde al 2, hasta el 1001 que es el 9. Los números entre el 11 y el 15 (1010, 1011, 1100, 1101, 1110, 1111) no se utilizan en el sistema decimal codificado en binario. En su lugar, 10 (decimal) toma los siguientes 4 bits (10 es 0001 0000 en BCD). Un byte (8 bits) representa a dos dígitos en decimal.

0	0000	0000	10	0001	0000	....	90	1001	0000
1	0000	0001	11	0001	0001	....	91	1001	0001
2	0000	0010	12	0001	0010	....	92	1001	0010
3	0000	0011	13	0001	0011	....	93	1001	0011
4	0000	0100	14	0001	0100	....	94	1001	0100
5	0000	0101	15	0001	0101	....	95	1001	0101
6	0000	0110	16	0001	0110	....	96	1001	0110
7	0000	0111	17	0001	0111	....	97	1001	0111
8	0000	1000	18	0001	1000	....	98	1001	1000
9	0000	1001	19	0001	1001	....	99	1001	1001

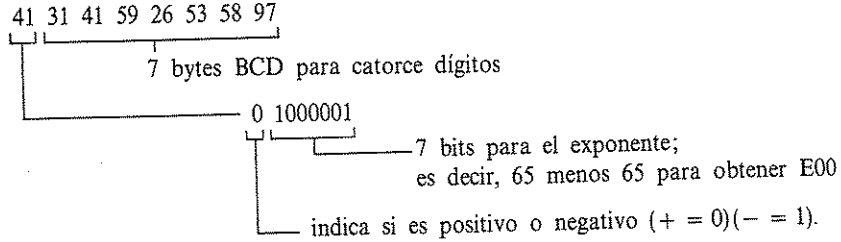
Número de 8 bytes de doble precisión en BCD

El MSX almacena los datos numéricos en doble precisión, a menos que esté programado de otra forma. Tiene una exactitud de catorce dígitos, y están almacenados en 8 bytes en el sistema decimal codificado en binario. El primer byte almacena el exponente, y los otros bytes dan los catorce dígitos; esto es, dos dígitos por byte.

Para ver cómo se almacenan los datos numéricos, primero debes encontrar la posición numérica de los datos utilizando la función VARPTR, y entonces saca el contenido (PEEK) y visualízalo en hexadecimal

```
10 PI=3.141592653489#
20 DIREC=VARPTR(PI)
30 FOR I=DIREC TO DIREC+7
40 PRINT HEX$(PEEK(I));" . ";
50 NEXT I
60 PRINT
run
```

Vamos a analizar lo anterior:



El byte de la cabeza nos da el signo, positivo o negativo, y el exponente. Para obtener el valor del exponente primero ignora el bit 7 y calcula el valor decimal para el resto (o sea, bits del 6 al 0); entonces réstale 65. Esto da un número cuyo tamaño está entre 1E-64 y 9,99999999999999E62.

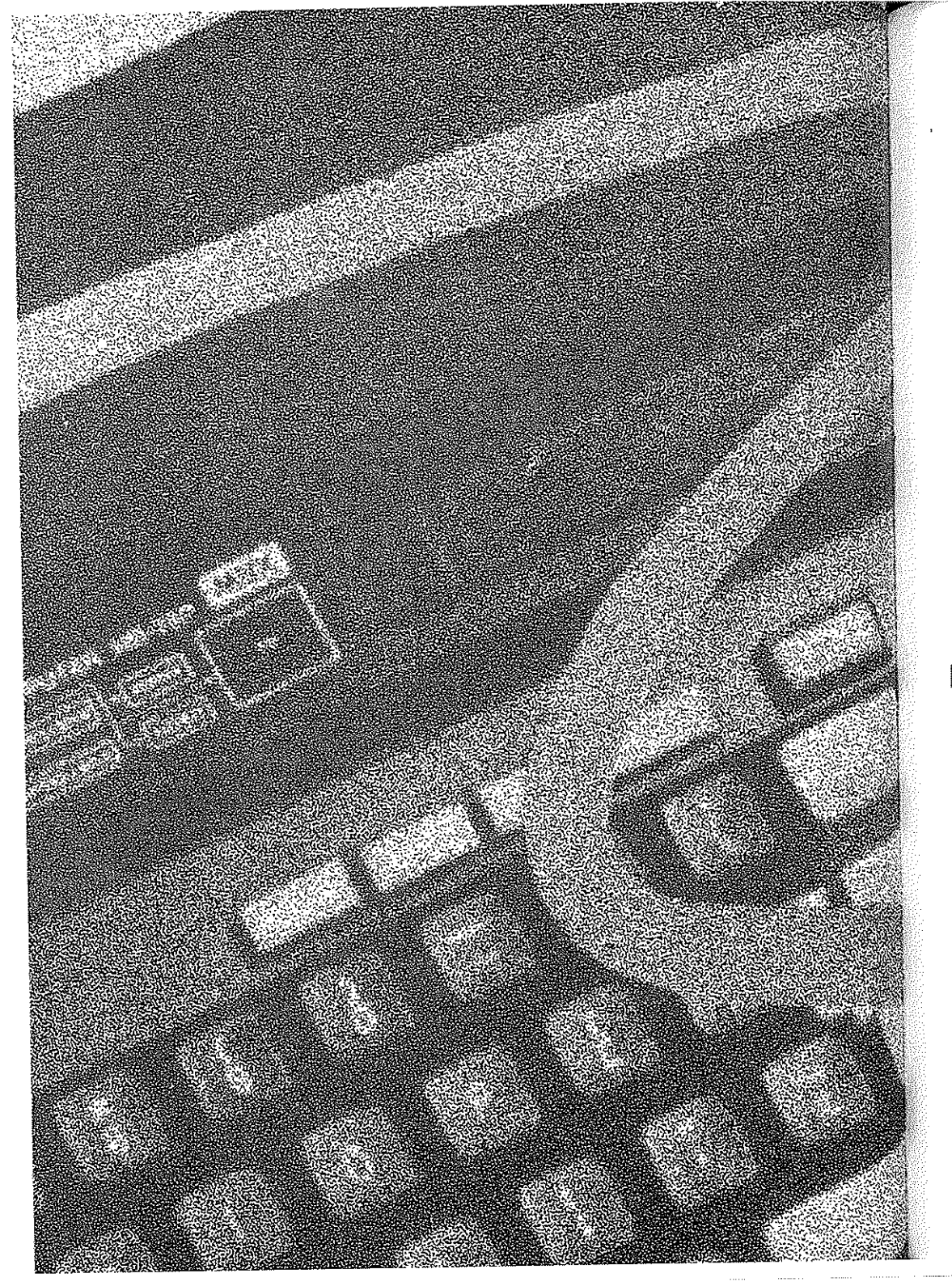
Ventajas y desventajas del sistema decimal codificado en binario

Ventajas

1. El BCD no da errores de redondeo, y puede dar números que no son posibles en binario, como 0,1.
2. Es extremadamente fácil visualizar un número en BCD porque no se necesita convertir de binario a decimal. Esto agiliza el proceso de impresión en la pantalla.

Desventajas

1. Al no estar en binario ordinario, el ordenador necesita más tiempo cuando está haciendo cálculos aritméticos. Sin embargo, el Z80 CPU tiene una instrucción especial DAA (acumulador de ajuste decimal), para hacerlo más fácil de procesar. El DAA convierte el contenido del acumulador en BCD empaquetado, siguiendo adición o sustracción, con los operandos del BCD empaquetado.



34

Algebra de Boole (I): operadores lógicos

Introducción

Aunque pequeño, tu MSX es un ordenador, y un ordenador está básicamente compuesto de millones de interruptores. Estos pueden estar en ON u OFF, y sus combinaciones llevan a cabo el trabajo del ordenador. La lógica del ordenador, o si prefieres, su gran cantidad de interruptores se gobierna por una serie de reglas que se engloban dentro del álgebra de Boole. Se utiliza principalmente en múltiples condiciones, en sentencias IF/THEN, con AND y OR, pero puedes emplearlo para tratar un solo bit, muy importante en el código máquina.

Operadores lógicos del MSX

El MSX posee una amplia gama de operadores booleanos (lógicos), a diferencia de algunos micros de 8 bits que parecen inhibirse de esta característica. Son muy importantes en la programación avanzada. Aquí tienes una lista por orden de precedencia.

1	NOT	complemento
2	AND	(AND) Y lógico
3	OR	(OR) O lógico
4	XOR	(OR) O exclusivo

5	EQV	Equivalente
6	IMP	Implicación

NOT, AND y OR son los operadores lógicos más importantes, ya que generalmente son los más utilizados, así como XOR, EQV e IMP, que pueden obtenerse por combinación de los operadores NOT, AND y OR.

Los operadores lógicos se aplican siempre a números enteros. Cualquier argumento que no sea entero se convierte en un número de 16 bits, los negativos, mediante su complemento a dos.

Ahora veremos cómo trabajan y cómo se pueden utilizar uno a uno, empezando con NOT

NOT

NOT es una negación. Cambia cierto por falso y viceversa; esto es, 1 por 0 y 0 por 1. En otras palabras: niega el argumento.

Aquí está la tabla de verdad del operador lógico NOT:

NOT	X	NOT X
	0	1
	1	0

NOT X	100	0000000001100100
	−101	1111111110011011

AND

El álgebra de Boole de la operación AND se puede listar en una tabla de verdad como ésta:

AND	X	Y	X AND Y
	0	0	0
	1	0	0
	0	1	0
	1	1	1

Por tanto, 100 AND 50 se puede calcular como sigue:

X	100	0000000001100100
AND Y	50	0000000000110010
	32	0000000000100000

Ejemplo: Uso del AND

La operación lógica AND se puede utilizar para conocer un bit en un número particular. Por ejemplo, si Y = 2^n, donde *n* es el bit a conocer entre 0 y 7;

entonces, si el bit de prueba *n* es 1, X AND Y = Y; y si el bit de prueba *n* es 0, X AND Y = 0.

El TEST del quinto bit (n = 5) para X = 117.

Y = 2^5 = 32		
X	117	0000000001110101
AND Y	32	0000000000010000
	32	0000000000010000

Por tanto, el quinto bit del número 117 es 1; o sea, verdadero.

OR

Cuando X, Y o ambos son verdaderos, OR devolverá el valor verdad. A la operación lógica OR también se la llama suma lógica.

La tabla de verdad de OR es:

OR	X	Y	X OR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1

Ejemplo: 34 OR 67

X	34	0000000000100010
OR Y	67	0000000001000011
	99	0000000001100011

Ejemplo: Uso del OR

El O lógico se emplea para convertir una letra mayúscula en una letra minúscula; la diferencia entre el código ASCII de una mayúscula y el de una minúscula es constante, y vale 32(2^5).

Ejemplo: Cambia el carácter A (ASCII código 65) en a (ASCII código 97).

A	65	0000000001000001
OR 2^5	32	0000000000100000
a	97	0000000001100001

Este método es bastante flexible, porque si intentas convertir una letra minúscula en otra letra minúscula, no ocurrirá nada. Inténtalo.

XOR

XOR, OR exclusivo devuelve el valor (1) cuando X e Y son diferentes.

XOR	X	Y	X OR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	0

Ejemplo: 100 XOR 50

X	100	0000000001100100
XOR Y	50	0000000000110010
	86	0000000001010110

XOR puede calcularse utilizando NOT, AND y OR. Aquí tienes cómo se calcula:

X XOR Y = (X AND (NOT Y)) OR ((NOT X) AND Y)
Simplemente, XOR es X AND NOT Y OR NOT X AND Y

Prueba:

X	Y	NOT X	NOT Y
0	0	1	1
1	0	0	1
0	1	1	0
1	1	0	0

X	(NOT Y)	X AND (NOT Y)
0	1	0
1	1	1
0	0	0
1	0	0

(NOT X)	Y	(NOT X) AND Y
1	0	0
0	0	0
1	1	1
0	1	0

(X AND (NOT Y))	((NOT X) AND Y)	(X AND (NOT Y)) OR ((NOT X) AND Y)
0	0	0
1	0	1
0	1	1
0	0	0

Por tanto, X XOR Y = (X AND(NOT Y)) OR ((NOT X) AND Y)

Ejemplo: Uso del XOR

XOR tiene la única propiedad de que si utilizas un valor de XOR con otro valor dos veces, obtienes el valor original.

X XOR Y XOR Y = X

Probaremos 100 XOR 50 XOR 50 como ejemplo:

100 XOR 50		
X	100	0000000001100100
XOR Y	50	0000000000110010
100 Y	86	0000000001010110

(100 XOR 50) XOR 50		
X	86	0000000001010110
XOR Y	50	0000000000110010
	100	0000000001100100

Por tanto, 100 XOR 50 XOR 50 = 100

EQV

Esta es la función lógica de equivalencia. EQV devuelve el valor cierto cuando X e Y son iguales, y falso cuando son diferentes.

EQV	X	Y	X EQV Y
	0	0	1
	1	0	0
	0	1	0
	1	1	1

Ejemplo: 51 EQV 75

X	51	0000000000110011
EQV Y	74	0000000001001010
	−122	1111111110000110

Verifica la siguiente relación:

X EQV Y = (X AND Y) OR ((NOTX) AND (NOT Y))

Prueba:

X	Y	NOT X	NOT Y
0	0	1	1
1	0	0	1
0	1	1	0
1	1	0	0

X	Y	(X AND Y)
0	0	0
1	0	0
0	1	0
1	1	1

(NOT X)	(NOT Y)	((NOT X) AND (NOT Y))
1	1	1
0	1	0
1	0	0
0	0	0

(X AND Y)	((NOT X) AND (NOT Y))	(X AND Y) OR ((NOT X) AND (NOT Y))
0	1	1
0	0	0
0	0	0
1	0	1

Por tanto:

X EQV Y = (X AND Y) OR ((NOT X) AND (NOT Y))

IMP

La tabla de verdad de IMP es:

IMP	X	Y	X IMP Y
	0	0	1
	1	0	0
	0	1	1
	1	1	1

100 IMP 50 se puede calcular como sigue:

X	100	0000000001100100
IMP Y	50	0000000000110010
	−69	1111111110111011

Cumple la siguiente relación:

X IMP Y = (NOT X) OR Y

Prueba:

X	Y	NOT X.
0	0	1
1	0	0
0	1	1
1	1	0

(NOT X)	Y	(NOT X) OR Y
1	0	1
0	0	0
1	1	1
0	1	1

Por tanto X IMP Y = (NOT X) OR Y.

Hasta aquí llega el BASIC del MSX, pero el álgebra de Boole no se detiene ahí. Hay muchas otras funciones útiles como NOR y NAND que no están incluidas en el MSX, pero pueden simularse mediante los operadores NOT, AND, y OR. Recuerda que estas tres son las básicas en el álgebra de Boole, y pueden combinarse para dar otras.

Mira la tabla de abajo: dos variables binarias, X e Y, y cada una puede tener un valor, 1 ó 0. Existen cuatro combinaciones diferentes de 0 y 1, ya que 2^2 = 4. Pueden hacerse, pues, dieciséis combinaciones (o funciones), ya que 2^2^2 = 16. Lo que obtienes es una lista de todas las posibles combinaciones o tablas de verdad.

X	Y	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0	0	0	1	0	1	0	0	0	1	0	1	1	1	0	1	1
0	1	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	1
1	0	0	1	0	0	1	0	1	0	0	1	0	1	1	1	0	1
1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	0	1	1

Podrás reconocer las siguientes:

f2 es NOT
f5 es AND
f9 es OR
f13 es XOR
f11 es IMP
f14 es EQV

Estas son todas las operaciones lógicas del BASIC del MSX, ¿pero qué pasa con las otras? (Por ejemplo, f1 y f3, etc.)

En esta parte veremos los operadores lógicos que no están en el MSX. No se pueden utilizar en el MSX como están, sino con una expresión equivalente con NOT, AND, OR.

- f0 Es siempre una función falsa cualquiera que sea la combinación de X e Y. No se utiliza para cálculos.
- f1 Siempre vale X cualquiera que sea el valor de Y.
- f2 NOT X (véase NOT).
- f3 Siempre es Y cualquiera que sea el valor de X.
- f4 NOT Y (véase NOT).

f5 X AND Y (véase AND).
f6 X AND (NOT Y).
f7 (NOT X) AND Y.
f8 Función NOR. Las equivalencias en MSX son:
(NOT X) AND (NOT Y), o también
NOT (X OR Y) ... (Véanse las leyes de De Morgan.)
f9 X OR Y (véase OR lógica).
f10 ((NOT X) OR Y). Similar a IMP.
f11 X IMP Y (véase IMP). Similar a f10. También puede escribirse como:
X OR (NOT Y).
f12 Función NAND. Es la negación de AND (NAND es NOT AND):
NOT (X AND Y)
(NOT X) OR (NOT Y)
X NAND Y = (NOT X) OR (NOT Y) = NOT (X AND Y)
(Véanse las leyes de De Morgan más adelante.)
f13 X XOR Y. OR exclusiva (véase XOR). Otro modo de escribirla es:
(X AND (NOT Y)) OR ((NOT X) AND Y).
f14 X EQV Y. Equivalencia lógica.
X EQV Y = (X AND Y) OR ((NOT X) AND (NOT Y)).
f15 Siempre cierta para cualquier valor de X e Y.
No se utiliza.

NAND y NOR, ambas no incluidas en el BASIC del MSX, son funciones útiles; recuérdalas.

Algunas relaciones lógicas útiles

Hay una variedad de relaciones lógicas que pueden utilizarse en el álgebra de Boole. Puedes emplearlas para minimizar expresiones largas.

CON OR

1. X OR 0 = X
2. X OR X = X
3. X OR X OR X OR X OR ... = X
4. X OR (NOT X) = 1
5. X OR 1 = 1
6. X OR (X AND Y) = X
7. X OR ((NOT X) AND Y) = X OR Y

CON AND

1. X AND 0 = 0
2. X AND X = X

3. X AND X AND X AND X ... = X
4. X AND (NOT X) = 0
5. X AND 1 = X
6. X AND (X AND Y) = X AND Y
7. X AND ((NOT X) AND Y) = 0

Ejemplos:

Prueba los siguientes ejemplos en tu ordenador y observa si las relaciones anteriores son ciertas.

	RESPUESTAS
PRINT 145 OR 0	145
PRINT 167 OR 167	167
PRINT 167 OR 167 OR 167	167
PRINT 133 OR (133 AND 222)	133
PRINT 111 AND 0	0
PRINT 111 AND 111	111
PRINT 111 AND 111 AND 111 ...	111
PRINT 234 AND (NOT 234)	0
PRINT 234 AND (234 OR 54)	234

Leyes de De Morgan

Aparte de las relaciones dadas, el álgebra de Boole tiene otras dos reglas especiales para reducir la longitud de las expresiones lógicas:

1.ª **Negación del OR lógico**
La negación (NOT) del OR lógico es equivalente al AND lógico de las negaciones de las variables que componen el OR lógico. Esto se expresa como:
NOT (X OR Y) = (NOT X) AND (NOT Y)

2.ª **Negación del AND lógico**
La negación del AND lógico es equivalente al OR lógico de las negaciones de las variables que comprenden el AND lógico. Esto se expresa como:

NOT (X AND Y) = (NOT X) OR (NOT Y)

Leyes de reorganización

En matemáticas hay un número de leyes que te permiten evaluar y simplificar expresiones complejas. Estas son las propiedades conmutativa, asociativa y distributiva. El álgebra de Boole tiene leyes parecidas, como:

1.ª **Leyes conmutativas**
a) X OR Y = Y OR X
b) X AND Y = Y AND X

2.ª **Leyes asociativas**

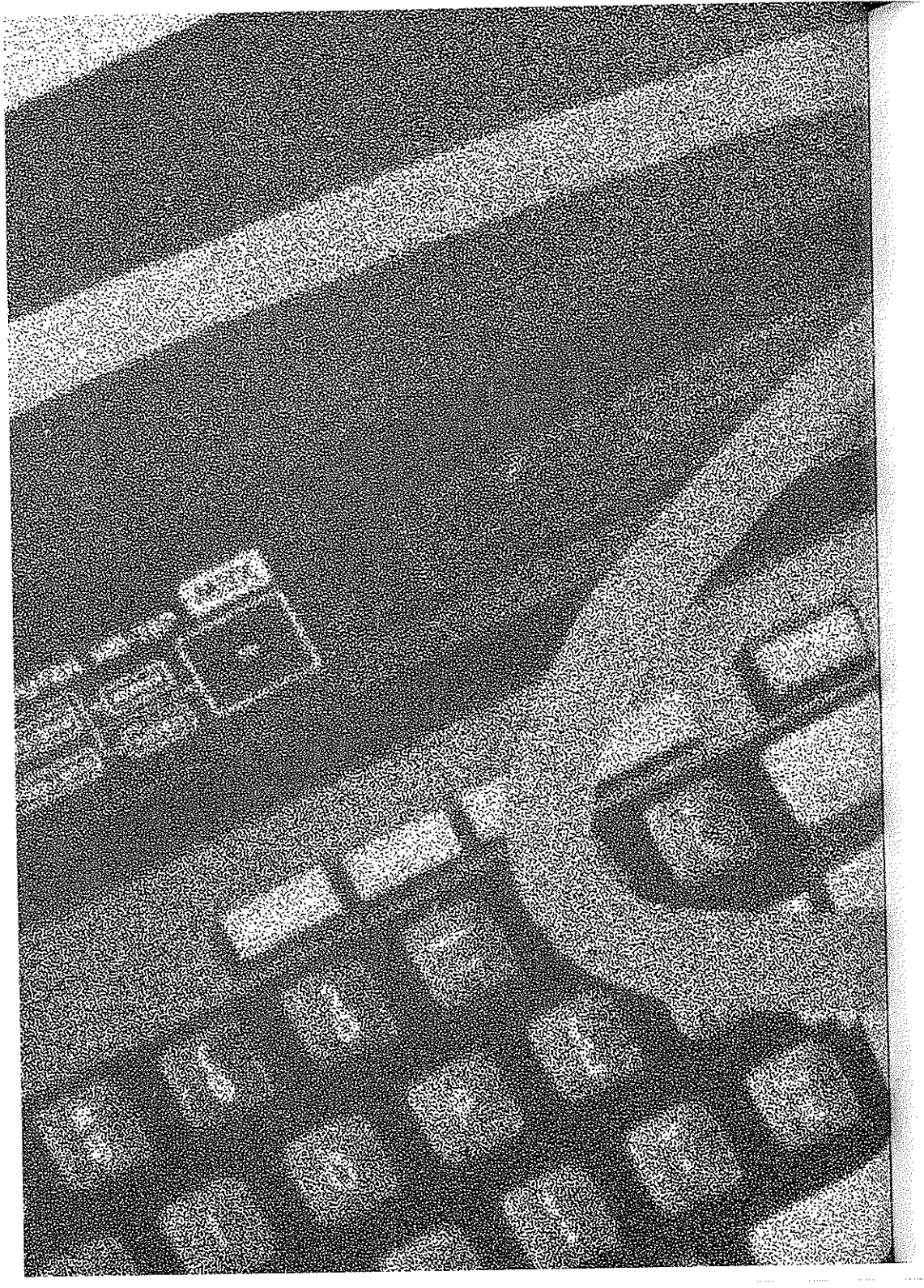
a) X AND (Y AND Z) = (X AND Y) AND Z = X AND Y AND Z
b) X OR (Y OR Z) = (X OR Y) OR Z = X OR Y OR Z

3.ª **Leyes distributivas**

a) X AND (Y OR Z) = (X AND Y) OR (X AND Z)
b) X OR (Y AND Z) = (X OR Y) AND (X OR Z)

Resumen de la tabla de verdad de los operadores lógicos del MSX

X	Y	AND	OR	XOR	EQV	IMP
0	0	0	0	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	0	1
1	1	1	1	0	1	1



35

Algebra de Boole (II): la instrucción IF/THEN/ELSE

Cuando escribas un programa es muy útil que el ordenador tome decisiones por sí mismo mientras tu programa se ejecuta. Para ello emplea la sentencia IF: primero pregunta sobre las condiciones y, dependiendo del resultado dado, llevará a cabo alguna tarea. IF se utiliza siempre en unión con THEN, y algunas veces también con ELSE. Tiene básicamente dos tipos de formatos:

IF <condiciones> THEN <sentencias>

e

IF <condiciones> THEN <sentencias> ELSE <sentencias>

IF va seguido por las <condiciones> que se tienen que analizar. Si el resultado de las <condiciones> es cierto, el ordenador ejecuta las <sentencias> que están detrás de la palabra THEN. Si el ordenador encuentra que las <condiciones> no son ciertas, entonces si no hay ELSE en la línea, seguirá a partir de la línea siguiente sin ejecutar las <sentencias> que hay después de THEN.

Sin embargo, si hay ELSE en esa línea, siendo las <condiciones> falsas, se ejecutan las <sentencias> después de ELSE. En este caso, las <sentencias> que hay entre THEN y ELSE se ignoran completamente.

Para empezar veamos la parte de <condiciones> de la estructura IF/THEN/ELSE.

Condiciones

Esta sección sigue todas las reglas descritas en los temas "expresiones y operadores" y "álgebra de Boole". Muestra la parte práctica de lo que dijimos en los dos últimos temas.

Comparación numérica

La condición más sencilla compara dos valores; por ejemplo, puede preguntar si un número es mayor que otro. Utiliza los operadores de relación, como <,>,<>,=,<=,>=.

Ejemplos:

1. 15 < 8 devuelve 0 (falso).
2. e% = 67 devuelve −1 (cierto si e% = 67
   0 (falso) si e% no es 67.

Y en formato IF THEN:

```
1. IF D% = 100 THEN PRINT "D% es 100"
2. IF V< = 6 THEN PRINT "V< = 6".
```

Estas son las formas más simples, pero puedes emplear expresiones aritméticas a ambos lados del operador de relación.

```
1. IF F%*H% = 9876 THEN PRINT "CIERTO"
2. IF 2^8 = N% – 19*Y% THEN PRINT "CIERTO".
```

No hay razón para que no se puedan utilizar funciones BASIC, como COS e INSTR, etc. De hecho puedes incorporar también funciones definidas por el usuario FN.

```
1. IF INSTR (A$,B$) = 5 THEN PRINT B$
2. IF COS(0.6242)> = TAN(D%) THEN PRINT "CIERTO"
3. IF FNA(D%*1% – 100) = &HFF THEN PRINT "FF"
```

Condición numérica sin comparación

No es estrictamente necesario tener una relación como <condición>. De hecho, puede ser una única variable numérica o una expresión sencilla. Mira este ejemplo:

```
IF X% THEN PRINT "X% = CIERTO"
```

En este caso el ordenador toma la condición como cierta cuando la variable X% sea distinta de 0, y falsa cuando X% = 0.

En general:

IF <expresión> THEN <sentencia>
<expresión> = CIERTA cuando <expresión> devuelve algo distinto de 0.
<expresión> = FALSA cuando <expresión> devuelve 0.

Puedes utilizarlo en una variedad de situaciones. Aquí tienes algunos ejemplos para ilustrar este punto.

1. IF A <> 0 THEN PRINT "A DISTINTO DE 0"
   ... puede ser
   IF A THEN PRINT "A DISTINTO DE 0"
2. Para comprobar si una cadena contiene un cierto carácter.
   IF INSTR (S$, "a") THEN PRINT "S$ CONTIENE EL CARACTER a".

Comparaciones de cadenas de caracteres

Puedes comparar dos cadenas de caracteres utilizando los operadores de relación casi de la misma forma. La comparación se efectúa cogiendo un carácter cada vez de cada cadena y comparando sus códigos ASCII. Si los códigos ASCII difieren, el número de código más pequeño precede al más grande. Para ver el código de cada carácter, véase la tabla de códigos ASCII. Si mientras se están comparando cadenas se llega al final de las mismas, la cadena más corta se dice que es la más pequeña. La comparación de cadenas de caracteres puede utilizarse para ordenar alfabéticamente.

En el ejemplo siguiente, todas las condiciones son ciertas.

1.ª) "abc"="abc"
2.ª) "ABC"<"abc"
3.ª) "ABC"<"ABD"
4.ª) "ab"<"abc"
5.ª) "1234"<"12345"

Condiciones múltiples empleando operadores booleanos

Es posible preguntar por varias condiciones de una vez, utilizando los operadores lógicos. Hay seis operadores lógicos en el BASIC del MSX, pero sólo tres de ellos, NOT, AND y OR, son relevantes. Siguen todas las reglas que gobiernan el álgebra booleano, así como las leyes de De Morgan.

1. Utilización sencilla de AND.
   IF <condición 1> AND <condición 2> THEN <sentencias>.

En este caso, sólo si ambas condiciones son ciertas el ordenador ejecutará las <sentencias> que siguen a THEN. Por ejemplo:

```
IF X = 0 AND Y$ = "YES" THEN PRINT "BRAVO"
```

2. Utilización sencilla de OR.
   IF <condición 1> OR <condición 2> THEN <sentencias>.

En este caso, si cualquiera de las dos condiciones es cierta el ordenador ejecutará las sentencias que siguen a THEN. Por ejemplo:

```
IF X = 0 OR Y = 0 THEN PRINT "UNO DE ELLOS ES CERO"
```

3. Utilización sencilla de NOT.
   IF NOT <condición> THEN <sentencias>

Si la condición es cierta, entonces NOT <condición> devuelve el valor opuesto; esto es, falso, y viceversa. Por ejemplo:

```
IF NOT (V = U*8) THEN PRINT "V NO ES U*8"
```

Habrás observado que es innecesario emplear NOT en la relación anterior, cuando lo puedes reescribir como:

```
IF V<>U*8 THEN PRINT "V NO ES U*8"
```

Aquí tienes una lista con esas relaciones que tienen el mismo significado.

NOT (X=Y)	X<>Y
NOT (X<>Y)	X=Y
NOT (X<=Y)	X>Y
NOT (X>=Y)	X<Y
MOT (X>Y)	X<=Y
NOT (X<Y)	X>=Y

Por tanto, no malgastes la memoria del ordenador.
Lo que NOT puede es invertir el resultado de una expresión como:

```
IF NOT INSTR(A$ , "Y") THEN PRINT "Y no se encuentra en la cadena A$"
```

El ejemplo anterior es autoexplicativo.

Minimización. Las leyes de De Morgan

NEGACION DE OR LOGICO:
Las siguientes condiciones tienen, exactamente, el mismo efecto.

```
IF NOT (X = 9 OR Y = 10)
IF NOT (X = 9) AND NOT (Y = 10)
IF X<>9 AND Y<>10
```

NEGACION DE AND LOGICO:
Las siguientes condiciones tienen, exactamente, el mismo efecto.

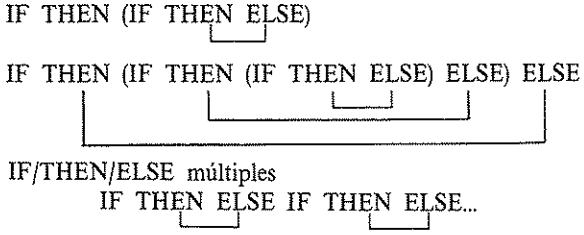
```
IF NOT (X = 9 AND Y = 10)
IF NOT (X = 9) OR NOT (Y = 10)
IF X<>9 OR Y<>10
```

Si lo dudas, compruébalas.

Estructuras IF/THEN/ELSE

ELSE es una parte de la estructura IF/THEN/ELSE. Indica al ordenador que, si la <condición> en IF no se satisface (es falsa), se saltan las sentencias que hay después de THEN y se efectúan las que hay después de ELSE.

Puedes programar múltiples IF/THEN/ELSE, así como entrelazarlos. Sólo estarán limitados por la longitud de una línea, que es de 255 caracteres. Si hay menos cláusulas ELSE que THEN, entonces cada ELSE se empareja al THEN más próximo.



La segunda sentencia IF se ejecuta después de que la primera sentencia IF sea falsa.

Sintaxis

IF ... THEN GOTO <línea> formato que puede ser abreviado de dos modos:

1. IF ... THEN <línea>
2. IF ... GOTO <línea>

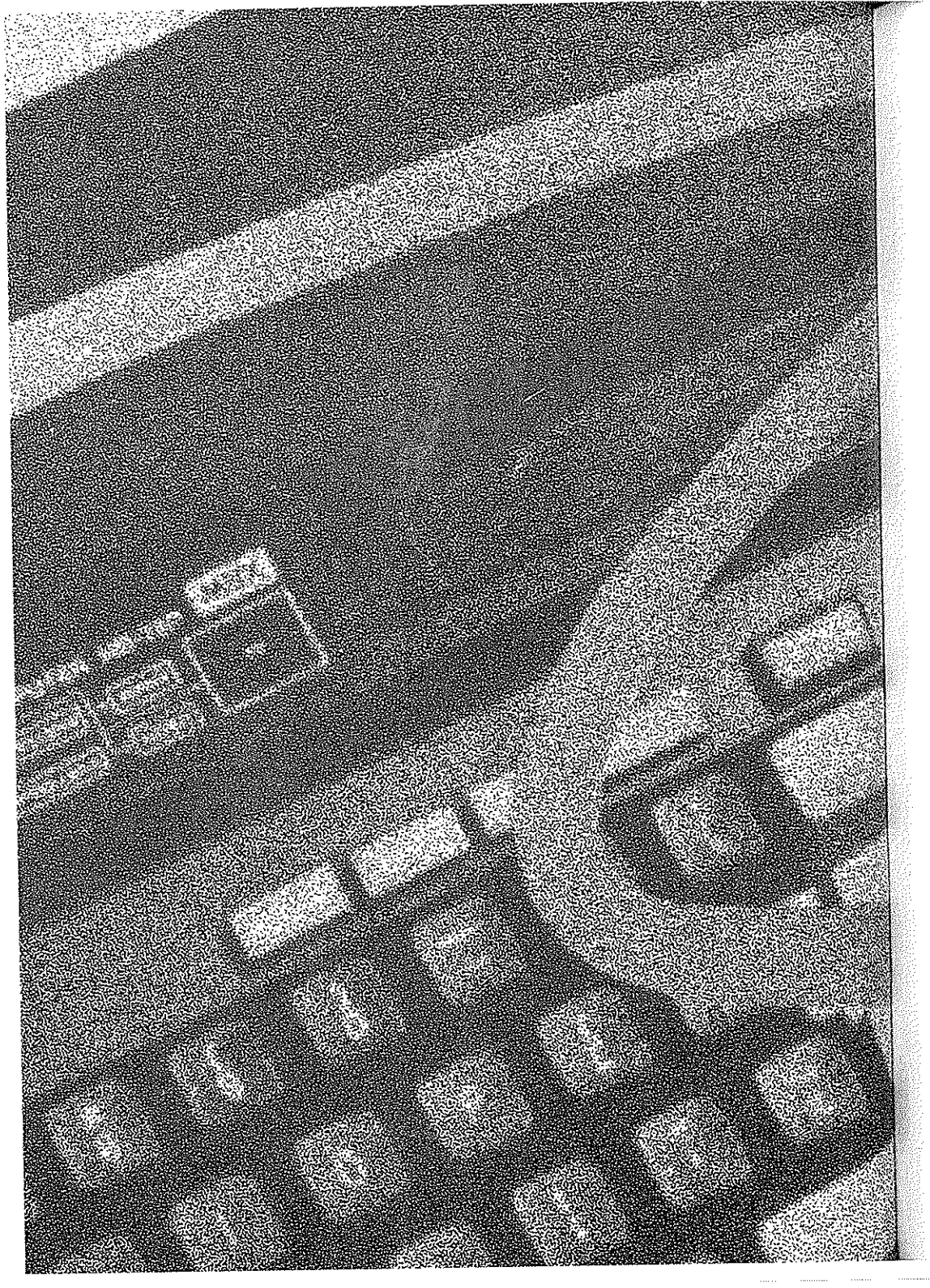
También ELSE GOTO <línea> puede ser abreviado:

1. IF ... THEN ... ELSE <línea>
2. IF ... THEN ... GOTO <línea>

Puntos a recordar

Las sentencias IF/THEN/ELSE tienden a ser muy largas para una sola línea, ya que pueden tener múltiples sentencias, después de THEN y ELSE. En este caso sería una buena idea emplear subrutinas mediante la sentencia GOSUB.

Ten cuidado con las sentencias IF/THEN/ELSE anidadas, ya que pueden producir desorden. Puedes crear un "programa espagueti" si no atiendes a ello...



36
PRINT USING

Introducción

El MSX tiene un completo control de formato para imprimir, a través de la sentencia PRINT USING. La sentencia PRINT USING permite imprimir caracteres y números en varios formatos.

Sintaxis

PRINT USING <car-exp>;<lista de elementos>.

El argumento de PRINT USING tiene dos partes, el formato específico de los caracteres y la lista de los elementos que se tienen que imprimir. Ambas están separadas por punto y coma. La <lista de elementos> pueden ser números o cadenas de caracteres. <car-exp> contiene el formato especial que determina cómo se tienen que imprimir los elementos. Puede ser tanto una cadena como una variable de tipo cadenas de caracteres.

Explicación de los caracteres del formato

! La exclamación especifica que tiene que imprimir el primer carácter de una cadena.

Aquí tienes un programa ejemplo, que imprime sólo la inicial de los nombres y apellidos dados en las sentencias de datos.

```
10 FOR I=1 TO 3
20 READ NOM$,APE$
30 PRINT USING "!";NOM$,".",APE$
40 NEXT I
50 DATA JUAN CARLOS,CONTRERAS
60 DATA CELSO,LOSADA
70 DATA JUAN JOSE,RODRIGUEZ

RUN
J.C
C.L
J.R
```

@ Inserta una cadena de caracteres dada en la posición indicada por @

```
10 A$ = "QWERTY"
20 PRINT USING "ESTO ES UN TECLADO @";A$

ESTO ES UN TECLADO QWERTY
```

\# El signo "#" indica que tiene que imprimirse un dígito en la posición especificada. Por ejemplo: PRINT USING "#.###"; 1.3. dará como resultado 1.300.

O sea, formatea el número que se visualice.

Como puedes ver en el ejemplo anterior debes incluir un punto decimal en la cadena del formato entre los signos "#". Si el número tiene menos dígitos que los especificados por los caracteres del formato, se ajustará por la derecha precedido de espacios.

```
10 PRINT "####.########"
20 PRINT USING "####.########"; ATN(1)*4

####.########
   3.1415927
```

Si los caracteres del formato especifican más espacios para decimales que los del número que se va a visualizar, los espacios vacíos se llenarán con ceros.

```
10 PRINT "##.#######"
20 PRINT USING "##.#######";99.999

##.#######
99.999000
```


Los números se redondearán si no caben en el espacio especificado.

```
10 PRINT "##.###"
20 PRINT USING "##.###";10.2367

##.###
10.237
```

Puedes imprimir dos o más números con el mismo formato en una misma sentencia PRINT USING.

```
10 PRINT USING "##.##    ";9.866,18.7

 9.87    18.70
```

Si el número que vas a imprimir es negativo, entonces el signo negativo se visualizará, pero ocupará el espacio de un dígito del formato especificado.

```
10 PRINT "###.####"
20 PRINT USING "###.####";-2.63

###.####
 -2.6300
```

Si el número que se quiere imprimir no cabe en el campo especificado por los caracteres del formato, se imprimirá el símbolo de tanto por ciento delante del número. Esto también sucede cuando el número redondeado excede el tamaño del campo.

```
10 PRINT USING "#.###";9.9999

%10.000
```

\+ Un signo más "+" al principio o al final del formato dará como resultado el signo del número que va a imprimirse, o sea, un "+" o un "−" en la posición especificada.

```
10 PRINT USING "+###.#####";-.123422,10.986

 -0.12342 +10.98600

10 PRINT USING "####.#+";10.71,100,-200.5

 10.7+ 100.0+ 200.5-
```

** El signo de doble asterisco provoca que los espacios en blanco del campo numérico se llenen con "*". Representan dos espacios de dígitos más el campo.

```
10 PRINT "**##.########"
20 PRINT USING "**##.########";ATN(1)*4
30 PRINT USING "**##.########";-ATN(1)*4

**##.########
***3.14159265
**-3.14159265
```

$$ El signo especial doble dólar pone un signo delante del número. $$ especifica dos espacios más en el campo, uno de ellos será el signo $ (dólar).

```
10 PRINT"$$+#.########"
20 PRINT USING "$$+#.########";ATN(1)*4
30 PRINT USING "$$+#.#####.##";-ATN(1)*4

$$+#.########
 $+3.141559265
 $-3.141559265
```

Ten presente que no puedes utilizar el formato exponencial en conjunción con el signo $$.

**$ Es una combinación de los signos ** y $$. Todos los espacios en blanco se llenan con *, y el número irá precedido por el signo $. El **$ especifica tres posiciones más para dígitos, uno de los cuales es el signo $.

```
10 PRINT " **$#.########"
20 PRINT USING "**$+#.########";ATN(1)*4
30 PRINT USING "**$+#.########";-ATN(1)*4

 **$#.########
**$+3.14159265
**$-3.14159265
```


La coma se sitúa a la izquierda del punto decimal en la especificación de formato , y motiva la aparición de una coma cada tres dígitos a la izquierda del punto decimal*.

```
10 PRINT"##########,.##"
20 PRINT USING"##########,.##";1090382.88#
30 PRINT"$$##########,.##"
40 PRINT USING"$$##########,.##";1090382.88#

##########,.##
  1,090,382.88
$$##########,.##
  $1,090,382.88
```

Una coma al final de la cadena de formato motivará la aparición de una coma al final del número.

```
10 PRINT USING "##.##,";12.567

12.57,
```

^^^^ Este es el formato especificado para exponentes. Dejando espacio para E+xx también puedes especificar la posición del punto decimal. Los dígitos significativos se ajustan a la izquierda, y el exponente se une a ellos.

```
PRINT USING"##.##^^^^";200.00

 2.00E+02

PRINT USING"+#.##^^^^";200.00

+2.00E+02

PRINT USING"#.##^^^^+";-200.00

2.00E+02-
```

Notas:

Si el número de dígitos especificados excede a 24 resultará un error de llamada ilegal a una función.

* El punto y la coma decimales anglosajones se emplean de forma inversa que nuestro punto y coma decimales. [*N. del T.*]

LPRINT USING

LPRINT USING es exactamente la misma instrucción que PRINT USING, pero escribiendo en la impresora.

PRINT# USING

PRINT# USING es también la misma instrucción que PRINT USING, pero escribe en la unidad especificada por el número de fichero que hay después de la marca #. Antes de ejecutar esta sentencia debes ejecutar una OPEN para abrir un canal para la unidad en que quieres escribir.

Sintaxis

PRINT# <número de fichero>, USING <car-exp>; <lista de expresiones>.

37

Sucesos e interrupciones en BASIC

Introducción

El ordenador MSX posee varias instrucciones para el tratamiento de sucesos. Estos sucesos se manejan mediante interrupciones. O sea, que cuando el ordenador está ejecutando un programa, al mismo tiempo está pendiente de que pueda ocurrir un suceso particular. En este caso se interrumpe inmediatamente el programa en curso y se salta a la subrutina determinada por el usuario.

El MSX puede tratar las interrupciones de los siguientes sucesos:

1. En intervalo de tiempo	ON INTERVAL GOSUB.
2. Pulsando las teclas de función	ON KEY GOSUB.
3. Pulsando <CTRL> <STOP>	ON STOP GOSUB.
4. Pulsando el disparador *<joystick>*	ON STRIG GOSUB.
5. Colisión de *sprites*	ON SPRITE GOSUB.
6. Errores	ON ERROR GOSUB.

1. Interrupciones de intervalo de tiempo

Las instrucciones que se emplean son:

ON INTERVAL = <tiempo> GOSUB <línea>.
INTERVAL ON/OFF/STOP.

El MSX tiene un reloj interno que se pone en funcionamiento cuando conectas el ordenador. Con la función TIME puedes saber el tiempo que lleva conectado. TIME se incrementa cada 1/50 de segundo, en que el procesador de video hace una interrupción.

Mediante la sentencia ON INTERVAL GOSUB especifica el intervalo de tiempo que ha de transcurrir para que haya una interrupción. Como el tiempo (TIME) se incrementa cada 1/50 de segundo, debes igualar a 50 el intervalo si quieres que la interrupción se produzca cada segundo; o sea, ON INTERVAL = 50 GOSUB <línea>.

ON INTERVAL = <tiempo> GOSUB también señala la subrutina a la que el BASIC tendrá que saltar cuando ocurra la interrupción. El ordenador saltará a esta subrutina tan pronto se haga la interrupción, siendo indiferente a la línea en que se encuentra ejecutando el ordenador.

El intervalo de interrupción se activa con INTERVAL ON. Esto indica al ordenador que comience la cuenta del reloj. Si esta instrucción no se ejecuta, la instrucción ON INTERVAL GOSUB no tendrá validez.

Una vez haya transcurrido el tiempo de interrupción se ejecutará automáticamente INTERVAL STOP. Esto evita que haya otra interrupción durante el tratamiento de la interrupción en curso. Sin embargo, si ha habido una interrupción durante la ejecución de la actual, el ordenador ejecutará de nuevo la subrutina; debes incluir la orden INTERVAL OFF en la subrutina para no meterte en un bucle de tratamiento de interrupciones sin fin.

Después de ejecutar la subrutina de interrupción, el ordenador ejecutará automáticamente INTERVAL ON para preparar una nueva interrupción, a menos que hayas incluido INTERVAL OFF en la subrutina de tratamiento.

Puedes inutilizar las interrupciones de tiempo cuando desees, utilizando INTERVAL OFF.

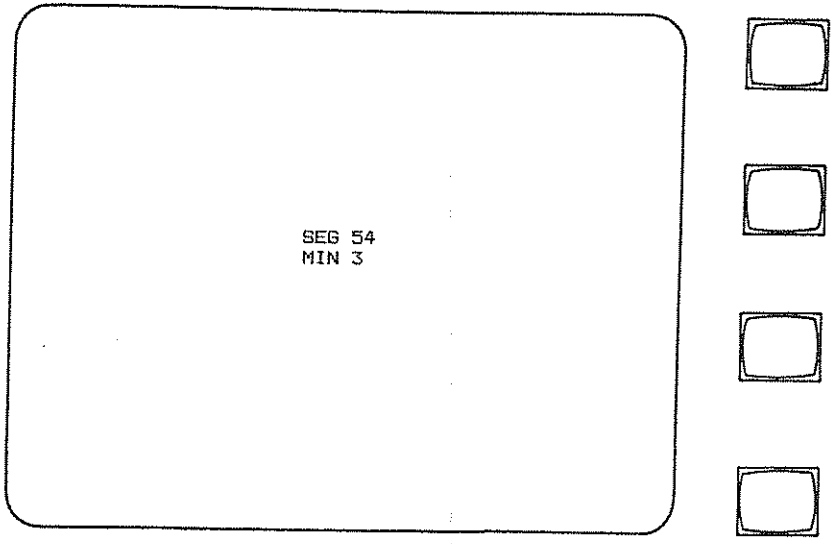
Las interrupciones temporales no tienen validez fuera del alcance de un programa BASIC; o sea, no son posibles trabajando en modo directo. Tampoco tienen validez en las subrutinas de tratamiento de errores.

Ejemplo:

Aquí tienes un programa corto que simula un reloj digital con sonido (beeps):

```
10  ON INTERVAL=50 GOSUB 60
20  INTERVAL ON
30  CLS
40  S=0: M=0
50  GOTO 50
55  REM Subrutina de intervalos
60  IF S=60 THEN S=0: M=M+1: LOCATE 10,11: PRINT "
MIN ";M: BEEP
70  S=S+1
80  LOCATE 10,10: PRINT "SEG ";S
90  BEEP
100 RETURN
```


LINEA 10 FIJA EL INTERVALO A 1 SEGUNDO PARA LA SUBRUTINA (60).
LINEA 20 ACTIVA LA DETECCION DE INTERRUPCIONES.
LINEA 30 BORRA LA PANTALLA.
LINEA 40 INICIALIZA S y M.
LINEA 50 BUCLE INFINITO PARA DETECTAR INTERRUPCIONES.
LINEA 60 SI TENEMOS 60 SEGUNDOS ENTONCES ES 1 MINUTO.
LINEA 70 S+1 SEGUNDOS.
LINEA 80 IMPRIME LOS SEGUNDOS.
LINEA 90 SONIDO (BEEPS).
LINEA 100 VUELVE A DONDE HABIA DEJADO EL PROGRAMA: EN ESTE CASO SIEMPRE ES A LA LINEA 50.



2. Interrupciones de las teclas de función

Las sentencias que se utilizan son:

ON KEY GOSUB <línea>,<línea>...
KEY (<número de función>) ON/OFF/STOP.

Es posible fijar interrupciones para las teclas de función mediante ON KEY GOSUB y las instrucciones KEY () ON/OFF/STOP.

ON KEY GOSUB va seguido por una lista de números de línea, especificando

las subrutinas de interrupciones para cada tecla de función. En caso de que haya una función sin un tratamiento específico, entonces se omitirá el número de línea. Las instrucciones KEY (<número de función>) ON activan una interrupción de la función especificada; a menos que se ejecuten estas sentencias, el ordenador no esperará una interrupción de la tecla de función no activada. Cuando haya una interrupción el ordenador saltará a la subrutina correspondiente especificada por la instrucción ON KEY GOSUB.

Una vez que se produce la interrupción de una tecla de función se ejecutará automáticamente KEY (<número de función>). STOP. Esto evita que se produzca otra interrupción de la misma tecla de función durante la ejecución de la subrutina de tratamiento de la interrupción. Sin embargo, recuerda si has pulsado esta tecla de función durante la subrutina, el ordenador volverá inmediatamente a la misma subrutina una vez haya ejecutado, a menos que la subrutina inutilice la interrupción de la tecla ejecutando KEY (<número de función>) OFF.

Después de abandonar la subrutina de tratamiento de la interrupción el ordenador ejecutará automáticamente KEY ON para esperar otra interrupción, excepto si hemos ejecutado KEY (<número de función>) OFF en dicha subrutina.

Las interrupciones de teclas de función se inutilizan cuando no esté ejecutando el programa, y también en las rutinas de tratamiento de errores.

Ejemplo:

```
10  ON KEY GOSUB 100,120,140,160,180,200
20  KEY (1) ON
30  KEY (2) ON
40  KEY (3) ON
50  KEY (4) ON
60  KEY (5) ON
70  KEY (6) ON
80  GOTO 80
90  REM Subrutinas
100 PRINT "FUNCION 1"
110 RETURN
120 PRINT "FUNCION 2"
130 RETURN
140 PRINT "FUNCION 3"
150 RETURN
160 PRINT "FUNCION 4"
170 RETURN
180 PRINT "FUNCION 5"
190 RETURN
200 PRINT "FUNCION 6"
210 RETURN
```

LINEA 10 APUNTA A LAS SUBRUTINAS DE INTERRUPCIONES CON F1, F2, F3, F4, F5 y F6.

LINEA 20 ACTIVA LA ESPERA DE UNA INTERRUPCION CON LA FUNCION F1.
LINEA 30 ACTIVA LA ESPERA DE UNA INTERRUPCION CON LA FUNCION F2.
LINEA 40 ACTIVA LA ESPERA DE UNA INTERRUPCION CON LA FUNCION F3.
LINEA 50 ACTIVA LA ESPERA DE UNA INTERRUPCION CON LA FUNCION F4.
LINEA 60 ACTIVA LA ESPERA DE UNA INTERRUPCION CON LA FUNCION F5.
LINEA 70 ACTIVA LA ESPERA DE UNA INTERRUPCION CON LA FUNCION F6.
LINEA 80 BUCLE INFINITO PARA ESPERAR LA INTERRUPCION DE UNA TECLA DE FUNCION.
LINEA 100 SUBRUTINA PARA F1.
LINEA 110 VUELVE A LA POSICION DE DONDE VINO.
LINEA 120 SUBRUTINA PARA F2.
LINEA 130 VUELVE A LA POSICIO4N DE DONDE VINO.
LINEA 140 SUBRUTINA PARA F3.
LINEA 150 VUELVE A LA POSICION DE DONDE VINO.
LINEA 160 SUBRUTINA PARA F4.
LINEA 170 VUELVE A LA POSICION DE DONDE VINO.
LINEA 180 SUBRUTINA PARA F5.
LINEA 190 VUELVE A LA POSICION DE DONDE VINO.
LINEA 200 SUBRUTINA PARA F6.
LINEA 210 VUELVE A LA POSICION DE DONDE VINO.

3. La interrupción <CTRL> <STOP> y el test de parada

Las sentencias que se emplean son:

ON STOP GOSUB <línea>
STOP ON/OFF/STOP

Detener la ejecución de un programa en BASIC es muy sencillo: basta con pulsar <CTRL> <STOP>. Esta operación detiene la ejecución inmediatamente y te sitúa de nuevo en el modo directo. Sin embargo, necesitará algunas veces hacer un test de parada en un programa, para prevenir que los usuarios de tu programa vean su contenido. Esto se hace detectando <CTRL> <STOP> con ON STOP GOSUB.

La sentencia ON STOP GOSUB fija una subrutina para las interrupciones <CTRL> <STOP>. Dentro de esta subrutina puedes situar un mensaje diciendo "No me cortes", o bien una sentencia RETURN para que el programa continúe ejecutando a partir del punto donde se produjo la interrupción.

STOP ON/OFF/STOP permite/imposibilita la interrupción de <CTRL> <STOP>. Cuando se ejecuta STOP ON, el BASIC examina si se ha pulsado <CTRL><STOP> cada vez que se ejecuta una nueva instrucción. Si se detecta un <CTRL> <STOP>, entonces el BASIC continúa la ejecución en la subrutina especificada por la sentencia ON STOP GOSUB incluida anteriormente en el programa.

Cuando se produce una interrupción se ejecuta automáticamente un STOP STOP. Esto inhibe esta interrupción durante la ejecución de la subrutina. Sin embargo, el ordenador memoriza si se ha pulsado otra vez un <CTRL> <STOP> durante la ejecución de la subrutina, y vuelve inmediatamente a la subrutina una vez la haya dejado, a menos que ésta desactive la interrupción <CTRL> <STOP> ejecutando STOP OFF. Después de abandonar la subrutina de interrupción, el ordenador ejecutará automáticamente STOP ON, para permitir otra interrupción, excepto si dentro de la subrutina se ha ejecutado un STOP OFF.

La única manera de salirse del test de parada del programa es efectuar un RESET en el ordenador. Acuérdate de grabar tus programas con test de parada antes de ejecutarlos con RUN.

Ejemplos:

Aquí tienes una rutina de test de parada para que la incluyas dentro de tus programas.

```
10    ON STOP GOSUB 10000
20    STOP ON

. .
. .

   ESCRIBE AQUI TU PROGRAMA

. .
. .
9999  REM Subrutina CTRL-STOP
10000 RETURN
```

LINEA 10 FIJA LA SUBRUTINA DE PARADA (STOP) EN LA LINEA 10000.
LINEA 20 ACTIVA LA DETECCION DE PARADA.
LINEA 10000 VUELVE AL LUGAR EN DONDE SE DETECTO LA INTERRUPCION.

Observa que ON STOP no detecta la detención del programa con la tecla STOP: sólo evitará que pares la ejecución del programa mediante <CTRL> <STOP>.

Si te limitas a pulsar la tecla STOP observarás que tu programa se detiene y aparece el cursor. Pero si pulsas la tecla STOP por segunda vez el programa continuará su ejecución.

La interrupción con <CTRL> <STOP> se desactiva cuando no se está ejecutando el programa y también en las subrutinas de error.

4. Interrupción del *joystick*

Las sentencias utilizadas son:
ON STRIG GOSUB <lista de números de línea>
STRIG (<n>) ON/OFF/STOP

El número del disparador, <n>, para cada mando es el siguiente:

0 = barra espaciadora.
1 = disparador 1 del *joystick* 1.
2 = disparador 1 del *joystick* 2.
3 = disparador 2 del *joystick* 1.
4 = disparador 2 del *joystick* 2.

Todo *joystick* compatible con el MSX tiene dos disparadores. La barra espaciadora también se considera un disparador.

ON STRIG GOSUB fija una subrutina de interrupciones del disparador de dicho mando. Puedes definir una subrutina para cada disparador incluyendo los números de línea de la subrutina de cada disparador.

Si un disparador concreto no tiene subrutina de interrupción debes omitirlo y poner una coma.

Por ejemplo:

ON STRIG GOSUB 10000, 200,,,

interrumpirá llamando a una subrutina cuando pulse

a) disparador 0 (barra espaciadora)

o

b) disparador 1 en el *joystick* 1

pero no hace nada si se pulsa cualquier otro disparador.

STRIG (<n>) ON activa la detección de una interrupción del disparador especificado. Sólo puedes activar un disparador por cada una de estas instrucciones.

Una vez haya llevado a cabo la interrupción se ejecuta automáticamente un STRIG (<n>) STOP. Esto inhibe las interrupciones durante la ejecución de la subrutina de dicho disparador. Sin embargo, el ordenador recuerda si se ha pulsado algún disparador durante la subrutina e irá inmediatamente a la subrutina correspondiente una vez haya acabado con la actual, a menos que en la subrutina actual se imposibilite la detección de interruptores de ese disparador. El ordenador ejecutará automáticamente un STRIG (<n>) ON una vez haya terminado la subrutina para permitir otra interrupción.

La interrupción STRIG no estará activada cuando el programa no se esté ejecutando, ni tampoco durante las rutinas de detección de errores.

Ejemplos:

Aquí tienes un programa muy corto que demuestra el funcionamiento de las interrupciones de los disparadores, examinando cada uno de los mandos *(joystick)*. El programa está escrito para que se puedan examinar ambos mandos *(joystick)*, así como la barra. Si pulsas la tecla <s> el programa terminará.

```
10  ON STRIG GOSUB 100,120,140,160,180
20  STRIG(0) ON
30  STRIG(1) ON
40  STRIG(2) ON
50  STRIG(3) ON
60  STRIG(4) ON
70  IF INKEY$="s" THEN END
80  GOTO 70
90  REM Subrutinas del disparador
100 PRINT "has pulsado la barra espaciadora"
110 RETURN
120 PRINT "disparador 1 del joystick 1"
130 RETURN
140 PRINT "disparador 1 del joystick 2"
150 RETURN
160 PRINT "disparador 2 del joystick 1"
170 RETURN
180 PRINT "disparador 2 del joystick 2"
190 RETURN
```

LINEA 10 FIJA LAS LINEAS DE TODAS LAS SUBRUTINAS DEL DISPARADOR.
LINEA 20 ACTIVA LA DETECCION DEL DISPARADOR 0.
LINEA 30 ACTIVA LA DETECCION DEL DISPARADOR 1.
LINEA 40 ACTIVA LA DETECCION DEL DISPARADOR 2.
LINEA 50 ACTIVA LA DETECCION DEL DISPARADOR 3.
LINEA 60 ACTIVA LA DETECCION DEL DISPARADOR 4.
LINEA 70 SI SE PULSA <s> ENTONCES ACABA.
LINEA 80 VUELVE A LA LINEA 70.
LINEA 90 RUTINAS DE LOS DISPARADORES.

5. Interrupción por choque de *sprites*

Véase la parte de *sprite*, gráficos avanzados, para más detalles.

6. Detección de errores

Véase la parte de tratamiento de errores para más detalles.

38
Tratamiento de errores

Mensaje de error

Cuando el BASIC del MSX encuentra un error mientras ejecuta un programa o un comando, visualiza un mensaje de error y para la ejecución. Aquí tienes una lista de los mensajes de error.

Código	Mensaje	Descripción
1	NEXT sin FOR *(NEXT without FOR)*	La variable de la sentencia NEXT no se empareja con la variable de la sentencia FOR anterior, o se encuentra una instrucción NEXT sin una sentencia FOR previa.
2	Error de sintaxis *(Syntax error)*	Faltas cometidas con los caracteres o una puntuación equivocada.
3	Return sin GOSUB *(Return without GOSUB)*	Se encuentra una sentencia RETURN sin haber ejecutado GOSUB previamente.
4	Fin de datos *(Out of DATA)*	Se ha ejecutado una sentencia READ cuando se han utilizado todos los datos de las sentencias DATA, o no hay sentencias DATA de donde leer (READ).
5	Llamada ilegal a una función *(Illegal function call)*	Se pasa un parámetro ilegal a una función matemática o de tipo de cadena de caracteres.

Código	Mensaje	Descripción
6	Desbordamiento *(Overflow)*	El número es demasiado grande para el tipo de variable, función o sentencia que lo sustenta.
7	Desbordamiento de memoria *(Out of Memory)*	La ejecución se sale de la memoria porque el programa es muy extenso; tiene demasiados FOR, demasiados GOSUB, demasiadas funciones o variables.
8	Número de línea no definido *(Undefined line number)*	No existe el número de línea referido en un GOTO, un GOSUB, una sentencia IF/THEN/ELSE o un comando DELETE.
9	Indice fuera de rango *(Subscript out of range)*	El índice de un elemento de la matriz sobrepasa el rango declarado en DIM, o el índice no es correcto.
10	Matriz ya dimensionada *(Redimensioned Array)*	Dimensionas una matriz ya existente utilizando DIM.
11	División por cero *(Division by zero)*	Se ha encontrado una división por cero en una expresión.
12	Ilegal en modo directo *(Ilegal direct)*	Una sentencia que no está permitida en modo directo.
13	Error de tipos *(Type mismacht)*	Se asigna un número a una variable de tipo cadena de caracteres o viceversa.
14	Sobrepasa el espacio de una cadena de caracteres *(Out of string space)*	Desborda el área de trabajo asignado a las cadenas de caracteres en la RAM del sistema.
15	Cadena de caracteres demasiado larga *(String too long)*	Se ha creado una cadena de caracteres de más de 255 caracteres de longitud.
16	Fórmula de cadena de caracteres muy compleja *(String formula too complex)*	Expresión de cadena de caracteres muy compleja para que la ejecute el ordenador.
17	Continuación imposible *(Can't continue)*	Intento de continuación en un programa que se ha parado por un error, o que acaba de ser editado o ha finalizado su ejecución en END.
18	Función de usuario no definida *(Undefined user function)*	Llamada a una función FN sin haberla definido antes con la sentencia DEF FN.
19	Error del dispositivo de E/S *(Device I/O Error)*	Error en la entrada/salida del cassette, impresora, etc.
20	Error de verificación *(Verify error)*	Es un error fatal sin posible remedio. CLOAD? ha detectado un error en la verificación del cassette.

Código	Mensaje	Descripción
21	No tiene RESUME *(No RESUME)*	Una rutina de tratamiento de errores no tiene sentencia RESUME cuando el ordenador la está esperando.
22	RESUME sin error *(RESUME without error)*	Encuentra una sentencia RESUME fuera de una rutina de error.
23	Error sin mensaje *(Unprintable error)*	Se ha encontrado un error que no tiene mensaje.
24	Falta un operando *(Missing operand)*	Una expresión contiene un operador sin un operando que la siga.
25	Desbordamiento de una línea *(Line buffer overflow)*	Una línea tiene demasiados caracteres para entrar en el *buffer*.
26-49	Error sin mensaje *(Unprintable error)*	Reservado para futuras extensiones.
50	Desbordamiento del campo *(Field Overflow)*	Una sentencia de campo (FIELD) abarca más bytes de los especificados por la sentencia OPEN indicando la longitud del registro de un fichero de acceso aleatorio, o estamos al final del *buffer* mientras hacemos una E/S secuencial (PRINT# INPUT#) a un fichero de acceso aleatorio.
51	Error interno *(Internal error)*	Mal funcionamiento de la máquina.
52	Error en el número de fichero *(Bad file number)*	Una sentencia o comando hace referencia a un fichero inexistente o fuera del rango indicado por MAXFILES.
53	Fichero no encontrado *(File not found)*	Una sentencia LOAD u OPEN hace referencia a un fichero inexistente.
54	Fichero ya abierto *(File already open)*	Intento de abrir un fichero que ya estaba abierto.
55	Intento de lectura después del fin de fichero *(Input past end)*	Se ejecuta una sentencia INPUT después de que se hayan acabado todos los datos del fichero. Emplea EOF para prevenir este error.
56	Error en el nombre del fichero *(Bad file name)*	Das un nombre ilegal a un fichero con las sentencias LOAD, SAVE u otras de E/S.
57	Sentencia directa dentro de un fichero *(Direct statement in file)*	Se ha encontrado una instrucción-directa mientras se está haciendo una operación de carga en memoria. La carga se ha acabado.
58	E/S secuencial *(Secuential I/0 only)*	Se pretende hacer un acceso aleatorio en un fichero secuencial.

Código	Mensaje	Descripción
59	Fichero no abierto *(File not OPEN)*	Se hace referencia a un fichero que no está abierto.
60-255	Error sin mensaje *(Unprintable error)*	No tienen códigos de error. Los puede definir el usuario.

Tratamiento de errores

El BASIC del MSX tiene muchas funciones y sentencias para ayudarte a depurar tu programa. Es normal incorporar una subrutina de tratamiento de errores a tu programa mientras lo estás depurando. Te ayudará a eliminar las faltas que haya durante la ejecución de prueba.

Antes de continuar con las rutinas de tratamiento vamos a ver las instrucciones del BASIC empleadas en estas subrutinas.

ERL nos da la línea con el error.
ERL es una variable reservada que contiene el número de línea en donde se ha encontrado el error del programa ejecutado.
ERR indica el número del código de error.
ERR contiene el número del código de error después de que el ordenador haya detectado un error en el programa. Tiene un valor entre 1 y 255. El significado de los errores asociados al número ERR se han expuesto anteriormente.

ERROR

Hay principalmente dos modos de emplear la sentencia ERROR:

1. Para simular la ocurrencia de un error. Una sentencia ERROR con un argumento entero hará que el ordenador detecte un error, terminará la ejecución e imprimirá el mensaje de error con el número de línea.
2. Para crear errores definidos por el usuario. La sentencia ERROR junto con la función de error ON ERROR GOTO te permitirán crear tus propios errores. (Te lo explicaremos más adelante.)

ON ERROR <línea>

ON ERROR GOTO permite conocer errores y especifica la línea a la que tiene que ir una vez se haya detectado un error. Al detectarlo se saltará a la línea especificada para ejecutar la subrutina de tratamiento del error, si éste se da durante la ejecución o en modo directo, o sea, fuera del programa.

RESUME

RESUME indica reanudar la ejecución del BASIC después de que un procedimiento de tratamiento de errores se haya llevado a cabo mediante la sentencia ON ERROR GOTO. Después de que el error se haya analizado, RESUME le dirá al ordenador que continúe con la ejecución de acuerdo con la siguiente sintaxis:

RESUME o RESUME 0
Reanuda la ejecución a partir de la instrucción que causó el error.
RESUME NEXT
Reanuda la ejecución a partir de la instrucción siguiente a la que causó el error.
RESUME <línea>
Reanuda la ejecución a partir de la línea especificada.

Rutinas de tratamiento de errores

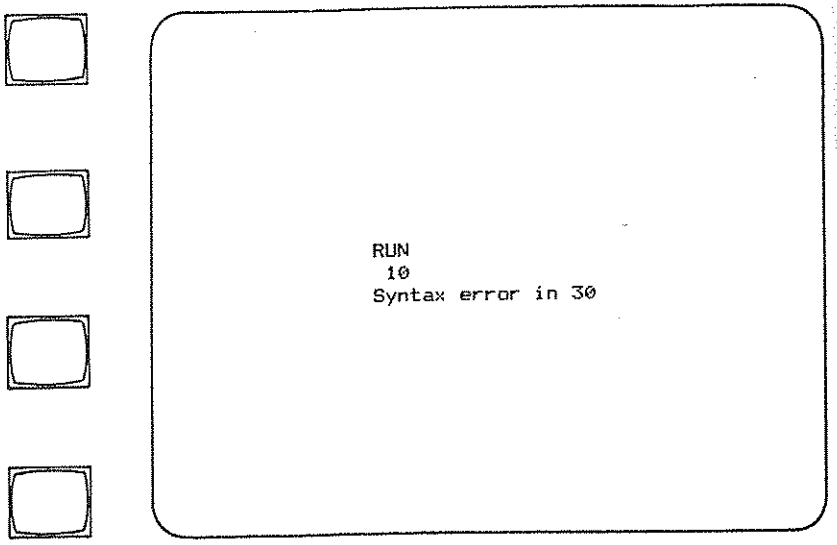
Para incorporar estas subrutinas en tu programa debes hacer lo siguiente:

1. Incluir la sentencia ON ERROR GOSUB justo al principio del programa para permitir la detección de errores a partir de ella. Después de que se haya ejecutado esta sentencia el ordenador buscará un error y la desplazará a la rutina de tratamiento de errores, ya estés en modo directo o en modo comando.
2. Una subrutina de tratamiento justo al final del programa. Aquí tienes un programa simple con una de estas rutinas.

```
10 ON ERROR GOTO 100
20 PRINT 10
30 PRONT 20
40 PRINT 30
50 END
99 REM Rutina de tratamiento de errores
100 ON ERROR GOSUB 0
```

LINEA 10 PERMITE LA BUSQUEDA DEL ERROR Y FIJA LA SUBRUTINA EN LA LINEA 100.
LINEA 30 EL ERROR ESTA AQUI.
LINEA 50 FINAL DEL PROGRAMA.
LINEA 99 ON ERROR GOSUB 0 HACE QUE EL BASIC SE PARE E IMPRIMA EL MENSAJE DEL ERROR.

El resultado de esta ejecución será:



Se ha detectado el error en la línea 30 y el programa salta a la línea 100. ON ERROR GOTO 0 muestra el error que ha causado la detención del programa. También desactiva la detección de errores.

Todo esto también puede hacerse sin este bucle de error. De no haber sentencia ON ERROR GOSUB el ordenador también detectaría el error en la línea 30 y dará el mensaje "Syntax error in 30".

Sin embargo, dicha rutina tiene otras funciones. Por ejemplo, la subrutina de tratamiento podrá tener un procedimiento para corregir el error y devolver el control al programa. Para ello necesitas saber qué tipo de errores deseas corregir y cómo hacer que el ordenador reanude la ejecución (RESUME).

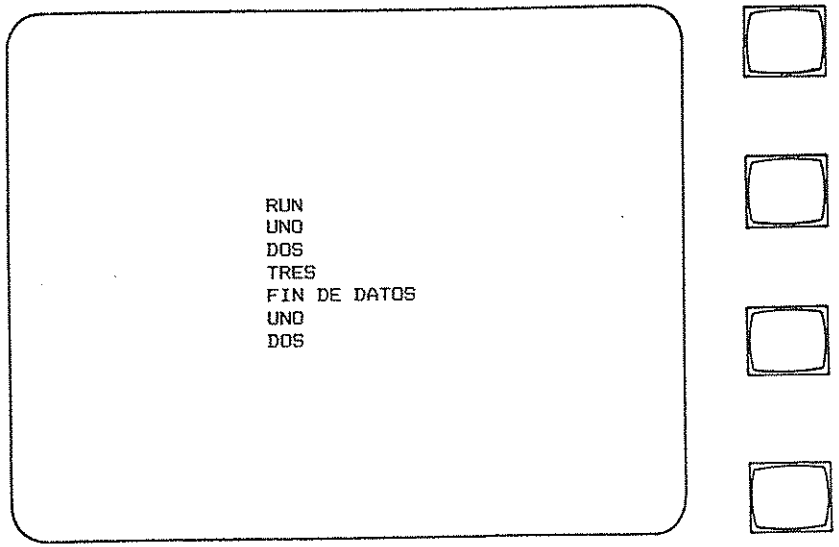
En este ejemplo no hay datos suficientes en la sentencia DATA de la línea 60. Ocasionando un error de falta de DATOS (código 4) en la cuarta ejecución del bucle. La rutina de tratamiento de errores tiene un procedimiento en la línea 100 para subsanar dicho error. Esta línea detecta si el error es por falta de datos, analizando el código de error ERR, y entonces ejecuta la instrucción RESTORE para que los datos puedan leerse de nuevo. Reanuda la ejecución (RESUME) a partir de la línea donde se detectó el error.

```
10  ON ERROR GOTO 100
20  FOR I=1 TO 5
30  READ A$
40  PRINT A$
50  NEXT I
60  DATA UNO,DOS,TRES
70  END
90  REM Rutina de errores
```



```
100 IF ERR=4 THEN RESTORE: PRINT "FIN DE DATOS":
RESUME
110 ON ERROR GOTO 0
```

LINEA 10 ACTIVA LA DETECCION DE ERRORES Y FIJA LA SUBRUTINA DE TRATAMIENTO DE ERRORES EN LA LINEA 100.
LINEA 20 HACE EL BUCLE CINCO VECES.
LINEA 30 LEE DATOS.
LINEA 40 IMPRIME A$.
LINEA 50 SIGUIENTE BUCLE.
LINEA 60 SOLO HAY TRES DATOS.
LINEA 70 END.
LINEA 100 SI ERR ES POR FALTA DE DATOS (CODIGO 4) ENTONCES HACER RESTORE Y CONTINUARA A PARTIR DE DONDE SE HABIA DEJADO.
LINEA 110 IMPRIME UN MENSAJE DE ERROR SI ENCUENTRA CUALQUIER OTRO ERROR.

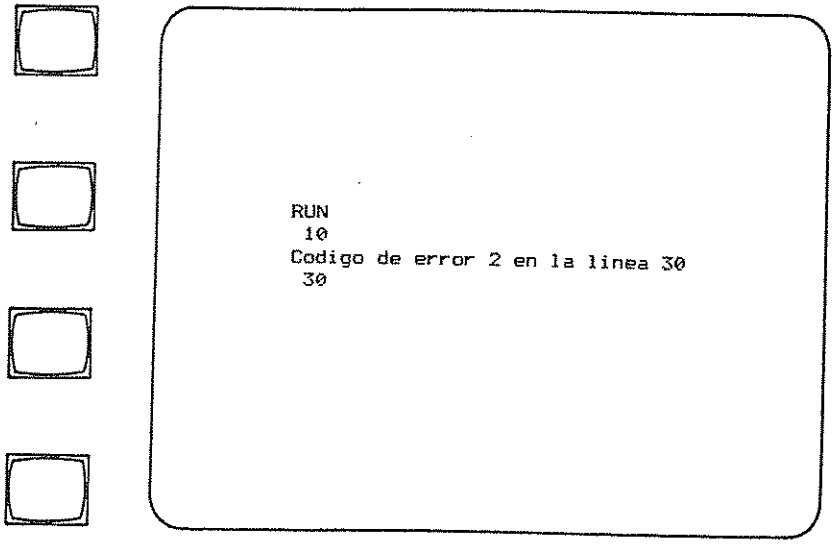


Con algunos errores quizá no quieras que el ordenador reanude la ejecución a partir de la misma línea. Errores como los sintácticos no pueden corregirse desde el programa: has de corregirlos mediante su edición. Sin embargo, si deseas que el ordenador salte la línea que tiene el error dándote un mensaje y continúe con el programa hasta el final emplea la sentencia RESUME NEXT, que reanuda la ejecución a partir de la línea siguiente a la del error.

```
10  ON ERROR GOTO 100
20  PRINT 10
30  PRONT 20
```

```
40  PRINT 30
50  END
99  REM Rutina de tratamiento de errores
100 PRINT "Codigo de error";ERR;"en la linea";ERL
110 RESUME NEXT
```

LINEA 10 ACTIVA LA DETECCION DE ERRORES Y FIJA LA SUBRUTINA DE ERROR EN LA LINEA 100.
LINEA 30 EL ERROR ESTA AQUI.
LINEA 50 FINAL DEL PROGRAMA.
LINEA 100 VISUALIZA EL CODIGO DE ERROR Y LA LINEA DEL ERROR.
LINEA 110 REANUDA LA EJECUCION A PARTIR DE LA LINEA SIGUIENTE.



```
RUN
 10
Codigo de error 2 en la linea 30
 30
```

Observa que puedes especificar la línea a la que salte en la sentencia RESUME. En el ejemplo anterior, RESUME 40 tendría el mismo efecto.

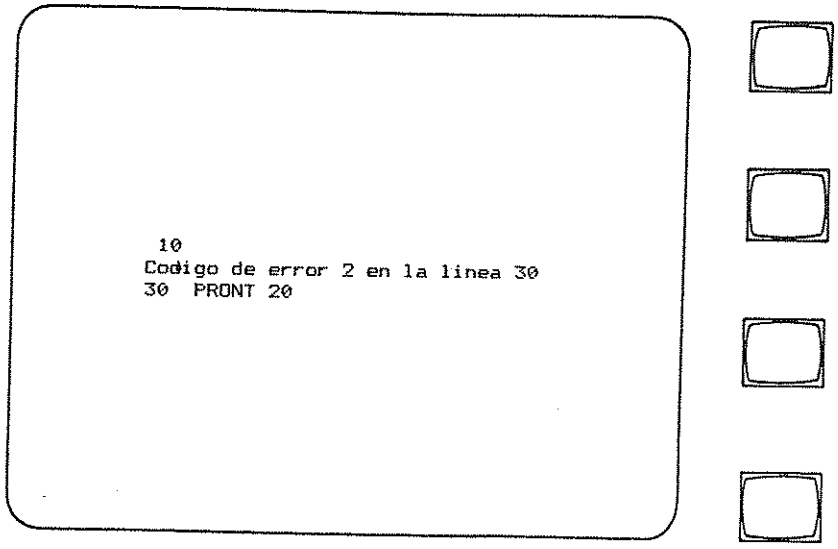
Es posible visualizar la línea que contiene el error desde la misma rutina de tratamiento. Será más sencillo corregir el error si la línea que lo contiene se visualiza nada más detectarlo. Para ello termina la rutina con la orden LIST. (LIST y un punto). El comando LIST. hará que se liste la última línea a la que se hizo referencia.

```
10  ON ERROR GOTO 100
20  PRINT 10
30  PRONT 20
40  PRINT 30
50  END
99  REM Rutina de tratamiento de errores
```



```
100 PRINT "Codigo de error";ERR;"en la linea";ERL
110 LIST.
```

LINEA 10 ACTIVA LA DETECCION DE ERRORES Y FIJA LA SUBRUTINA DE ERROR EN LA LINEA 100.
LINEA 30 EL ERROR ESTA AQUI.
LINEA 50 FINAL DEL PROGRAMA.
LINEA 100 VISUALIZA EL CODIGO DE ERROR Y LA LINEA DE ERROR.
LINEA 110 LISTA LA LINEA CON EL ERROR.



```
 10
Codigo de error 2 en la linea 30
30  PRONT 20
```

Ahora puedes corregir la línea 30 como:

30 PRINT 20

Cómo crear tus propios errores

Con la sentencia y la función de error ON ERROR GOTO podrás crear tus propios errores.

Hay unos treinta y seis errores con códigos entre 1 y 60 en la presente versión del MSX. Puedes emplear los números de código entre 61 y 255.

Para programar tus propios errores primero has de poner al ordenador en el modo de detección de errores, mediante la ejecución de la sentencia ON ERROR GOTO <línea> al principio del programa. Entonces, si en el programa surge una condición por la que quieres saltar a la rutina de error, hazlo con:

IF <condición> THEN ERROR <código de error>

Se activará así ON ERROR GOTO y el ordenador ejecutará inmediatamente la subrutina de tratamiento de errores. Dentro de la subrutina has de tener una línea que diga:

IF ERR = <código de error> THEN PRINT "Mensaje de error"

En el ejemplo siguiente todas las órdenes que incluyan la palabra MATAR se les tratará como un nuevo error (número 255). La rutina de error se vale de la función ERR.

```
10  ON ERROR GOTO 100
20  INPUT "MI AMO, DAME UNA ORDEN";A$
30  IF INSTR(A$,"MATAR") THEN ERROR 255
40  PRINT "VALE."
50  END
100 IF ERR=255 THEN PRINT "EL ASESINATO ESTA PENAL
IZADO EN ESTA AVENTURA.": RESUME 20
110 END
```

LINEA 10 ACTIVA LA DETECCION DE ERRORES.
LINEA 20 ENTRADA.
LINEA 30 COMPRUEBA SI LA PALABRA INVALIDA ESTA DENTRO DE LA ORDEN.
LINEA 100 MENSAJE DE ERROR. SE REANUDA A PARTIR DE LA LINEA 20.
LINEA 110 FINALIZA SI EL ERROR NO ES EL 255.

```
RUN
MI AMO, DAME UNA ORDEN?MATARLE
EL ASESINATO ESTA PENALIZADO EN ESTA AVENTURA.
MI AMO, DAME UNA ORDEN?PUES TE DESCONECTAS
VALE.
```


Es bastante útil que al definir tus propios códigos de error empieces a partir del 255 y vayas descendiendo; así evitarás chocar con futuras ampliaciones que los fabricantes de los ordenadores MSX hagan de los códigos de error dados.

Notas de errores

La sentencia ERROR puede emplearse también para simular la ocurrencia de un error. Por ejemplo:

ERROR 2

dara:

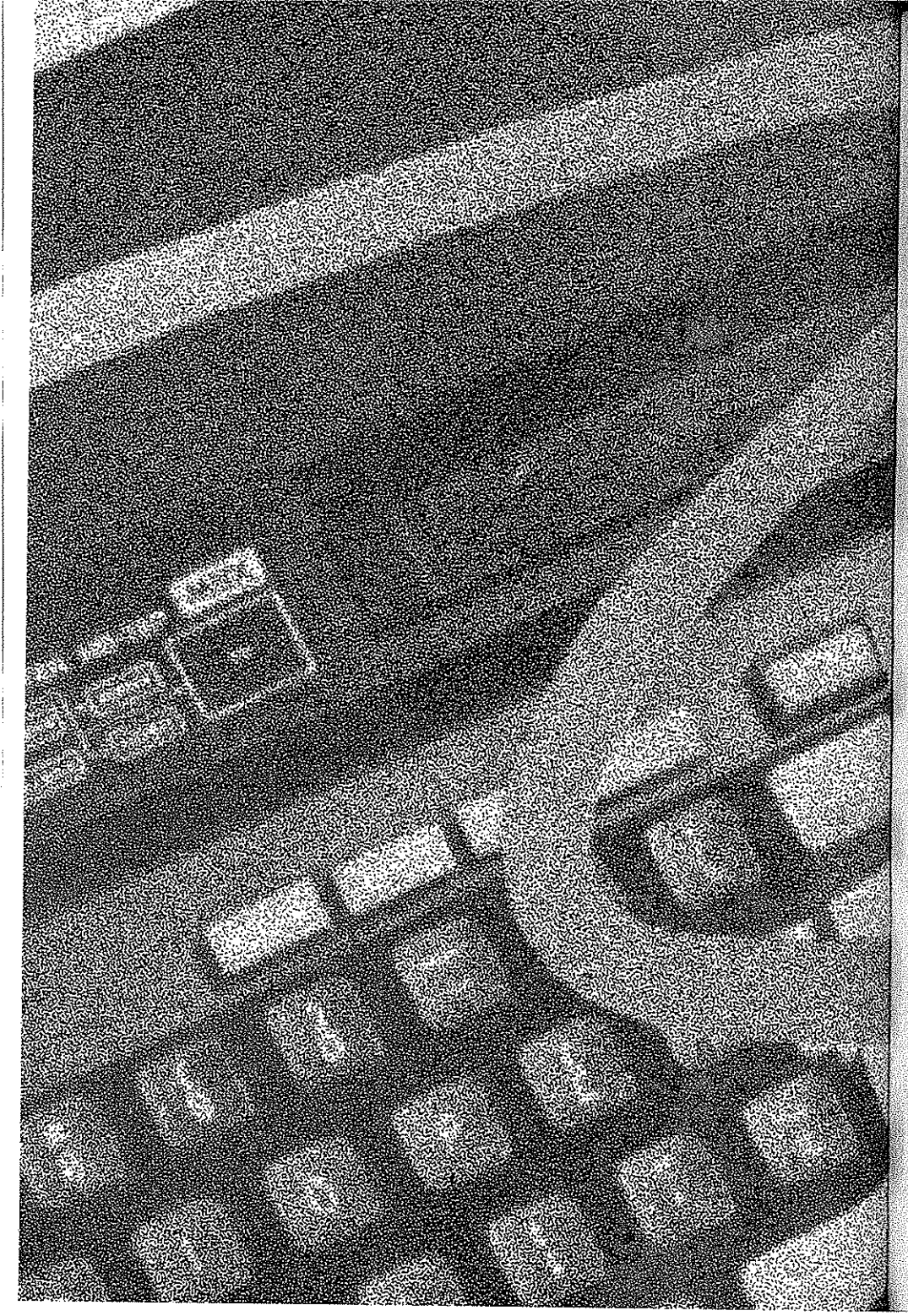
Syntax error (error sintáctico)

Si el código de error no tiene previamente definido un mensaje de error, cuando el ordenador lo detecte en el programa escribirá *Unprintable error* (error sin mensaje).

No se detectará ningún error durante la ejecución de la rutina de error. Si lo hay dentro de ésta, entonces se dará el mensaje de error correspondiente y se pasará el programa.

La rutina de tratamiento no puede sustentar todos los errores. Por ello, recomendamos terminar las rutinas de tratamiento de errores con ON ERROR GOTO 0, la cual detendrá la ejecución del programa mostrando el mensaje de error.

ON ERROR desactiva todas las detecciones de interrupción como ON INTERVAL y ON STRIG.



39

Grabación y carga con el cassette

Introducción

Hay varios modos para cargar y grabar programas y datos en un cassette. Esta sección muestra el manejo más sofisticado del cassette.

Instrucciones y funciones relacionadas con la carga y grabación del cassette

Aquí tienes una lista de instrucciones y funciones relacionadas con la pantalla.

Programas para salvar y cargar en BASIC (véase la introducción al MSX BASIC para más detalles):

CLOAD	Carga un programa BASIC del cassette.
CSAVE	Salva un programa BASIC al cassette.
CLOAD?	Compara un programa BASIC del cassette con el que está en la memoria del ordenador.
MOTOR	Pone en funcionamiento o desconecta (ON/OFF) el motor del cassette.

Para grabar, cargar y unir en formato ASCII:

SAVE	Graba un programa BASIC en una unidad especificada, con formato de fichero ASCII.

LOAD	Carga un programa BASIC grabado en formato ASCII del periférico especificado.
MERGE	Funde un programa BASIC salvado en una forma de fichero ASCII con el programa BASIC en la memoria del ordenador.

Grabar y cargar una sección de la memoria del ordenador como programa en código máquina o como datos:

BSAVE	Graba una sección de la memoria.
BLOAD	Carga una sección en la memoria.

Fija la velocidad de transmisión al grabar en cassette:

SCREEN
CSAVE

Velocidad de transmisión del cassette

Es la velocidad con que viajan los datos que son transmitidos al cassette. El MSX utiliza el formato FSK, que te dará la opción de dos velocidades: 1.200 y 2.400 baudios. Pueden fijarse tanto con la sentencia SCREEN como con el comando CSAVE. La velocidad de transmisión estándar es de 1.200 baudios. Su sintaxis es:

SCREEN ,,1	1.200 baudios
SCREEN ,,2	2.400 baudios
CSAVE "<nombre>"	1.200 baudios
CSAVE "<nombre>",1	1.200 baudios
CSAVE "<nombre>",2	2.400 baudios

La velocidad del cassette puede ser tanto de 1.200 como de 2.400 baudios, ya que CLOAD, LOAD y BLOAD percibirán automáticamente qué velocidad está empleando el cassette y cargarán de acuerdo con ella.

Grabación y carga en formato ASCII

Cuando un programa en BASIC se graba en el cassette, se hace en forma de *token*, ya que es el formato más eficiente. Sin embargo, si necesitas grabar tu programa en código ASCII debes utilizar la sentencia SAVE. Para cargar un programa grabado en código ASCII debes emplear la sentencia LOAD. En ambas, SAVE y LOAD, debes especificar el periférico en el cual estás grabando o del que vas a cargar mediante <perif-descrip>. Sin embargo, sólo se trabaja con el cassette en esta versión del MSX, por lo que el <perif-descrip> será siempre "CAS".

Sintaxis

SAVE "<perif-descrip> <nombre del programa>"	Salva un programa BASIC en un fichero ASCII.
LOAD "<nombre de perif>"	Carga el primer fichero BASIC que encuentre grabando en formato ASCII.
LOAD "<nombre de perif> <nombre del fichero>"	Carga el fichero BASIC especificado.
LOAD "<nombre de perif> <nombre del fichero>"R	Carga el fichero BASIC especificado y lo ejecuta.

Observa que la opción R en LOAD hará una autoejecución del programa BASIC nada más se haya cargado.

El interés principal de la grabación en formato ASCII es que el programa se puede unir a otro, que es lo que vamos a ver a continuación.

Unión de dos programas en BASIC

Supón que tenemos dos programas en BASIC: el programa 1 y el programa 2, y quieres unir el programa 2 al programa 1, por lo que el programa 2 vendrá después del programa 1 cuando eventualmente estén fundidos. Ambos están inicialmente grabados en el cassette.

PROG 1

```
10 REM PROGRAMA 1
20 PRINT "BIENVENIDO"
30 PRINT "AL"
40 PRINT "MSX"
```

PROG 2

```
10 REM PROGRAMA 2
20 FOR I=32 TO 58
30 PRINT CHR$(I)
40 NEXT I
```

Primero carga el programa 2 y renumera todas las líneas para que sean mayores que las del programa 1.

CLOAD "PROG 2"
RENUM 50
LIST

```
 50 REM PROGRAMA 2
 60 FOR I=32 TO 58
 70 PRINT CHR$(I)
 80 NEXT I
```

Entonces se graba en formato ASCII utilizando SAVE:

SAVE "CAS:PROG 2"

Carga el programa 1 del cassette:

CLOAD "PROG 1"

Después de que hayas cargado el programa 1, rebobina la cinta hasta donde esté grabado el PROG 2 y teclea:

MERGE "CAS:PROG2"

Pon en marcha el cassette. El ordenador unirá el programa 2 al final del programa 1.

```
 10 REM PROGRAMA 1
 20 PRINT "BIENVENIDO"
 30 PRINT "AL"
 40 PRINT "MSX"
 50 REM PROGRAMA 2
 60 FOR I=32 TO 58
 70 PRINT CHR$(I)
 80 NEXT I
```

Ten en cuenta lo siguiente cuando unas dos programas

1. Si tienes un número de línea igual en ambos, el programa inicial (1) y el programa secundario (2), el resultado de unir los programas contendrá la línea del segundo programa (2).
2. MERGE necesita que le indiques el periférico
   MERGE "<nombre del perif> <nombre del fichero>"
   <nombre del perif> = CAS: para el cassette.
3. Si se omite el nombre del fichero en la sentencia MERGE, entonces se unirá el siguiente fichero con formato ASCII que haya en la cinta.

Grabación y carga de una parte de la memoria del ordenador

Para grabar parte de la memoria localizada en una dirección específica en el cassette utiliza el comando BSAVE. Debes especificar las direcciones de comienzo y final de la parte con código máquina o datos.

Si está grabando un programa en código máquina puedes especificar la dirección de comienzo de la ejecución. Cuando cargues el código máquina con BLOAD, y si BLOAD ordena la autoejecución del programa en código máquina, ejecutará el programa desde la direccion de ejecución que especificaste en BSAVE.

Todo lo grabado con BSAVE debe cargarse utilizando BLOAD. Si se omite la dirección de ejecución al grabar BSAVE, entonces el ordenador asume que la dirección de ejecución es la dirección de comienzo si cargas el programa con BLOAD especificando la autoejecución con la opción R.

Sintaxis

BSAVE "<nombre del perif>:<nombre>",<dirección del comienzo>,<dirección final>.

BSAVE "<nombre del perif>:<nombre>",<dirección del comienzo>,<dirección final>,<dirección de ejecución>.

BLOAD "<nombre del perif>:<nombre>",R.
<nombre> y R son opcionales.

La opción R ejecuta automáticamente el programa en código máquina justo después de cargarlo. R significa RUN (ejecuta).

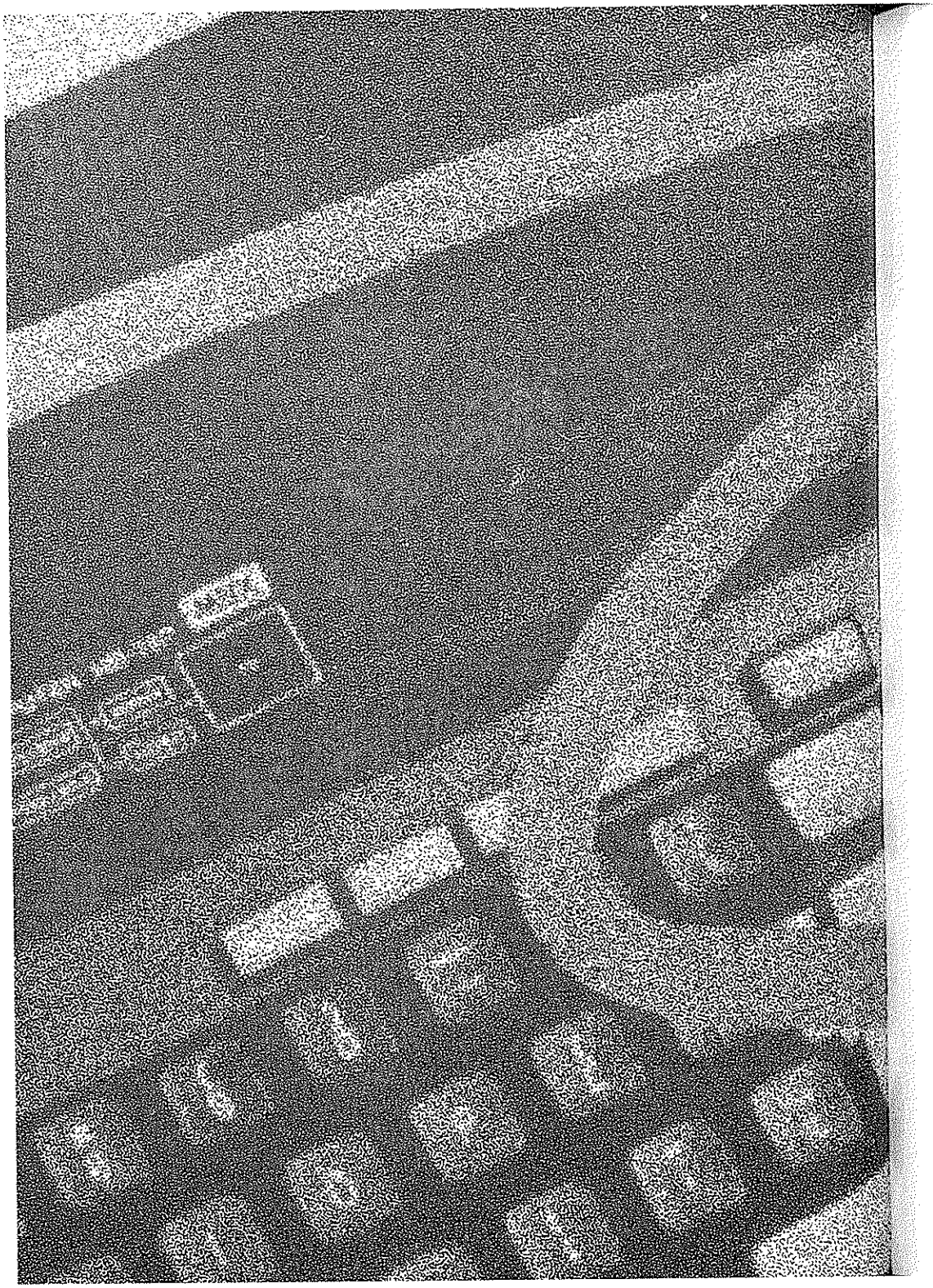
BLOAD "<nombre del perif>:<nombre>",<num-const>.

Cuando especificas <num-const> el fichero entrará en memoria a partir de la posición apuntada por <num-const>.

<nombre de perif> = CAS: para cassette.

La dirección se puede indicar en hexadecimal; o sea:

BSAVE "CAS:PROG",&HF300,&HF380,&HF30A.



40

Gráficos avanzados (I)

Características de cada modo de pantalla

El MSX posee uno de los *chips* gráficos más versátiles, el TMS 9929A, fabricado por Texas Instrument Inc., en Estados Unidos. Es uno de los más capacitados en el mercado y dispone de una gran variedad de facilidades, como poseer dieciséis colores para la alta resolución gráfica y animación de *sprites*. Puedes incluso manejar su propia RAM de 16K, con lo cual el procesador central se libera de asignar memoria para los gráficos de pantalla.

El BASIC del MSX ha sido implementado especialmente con estas características gráficas potentes y fáciles de programar para los principiantes. Su uso es sencillo; una vez hayas trabajado con él puedes crear dibujos espectaculares si lo intentas.

Vamos a empezar explicando los modos de pantalla que el MSX pone a tu alcance:

MODO DE PANTALLA

Modo	Texto/gráficos	Resolución	Color	Input	Gráficos	Sptrites
0	40 × 24 modo-texto	40 × 24 car.	2 de 16	SI	NO	NO
1	34 × 24 modo-texto	32 × 24 car.	2 de 16	SI	NO	SI
2	Hi-res gráfico	256 × 192 car.	16	NO	SI	SI
3	Multicolor	64 × 48 bloques	16	NO	SI	SI

Notas:

INPUT: Utilización o no de la sentencia INPUT en el modo dado de pantalla. No se puede emplear INPUT mientras nos encontramos en el modo gráfico.

GRAFICOS: Empleo de las sentencias gráficas, por ejemplo DRAW, en el modo de pantalla dado. Generalmente no podrás utilizar las sentencias gráficas en los modos de texto (0 y 1).

Sentencias y funciones relacionadas con el modo PANTALLA

Aquí hay una lista de sentencias y funciones que están relacionadas con la pantalla. Se dan más detalles en la parte de referencia del BASIC.

Sentencias relacionadas con todos los modos de PANTALLA

SCREEN	Fija el modo de pantalla.
CLS	Limpia la pantalla.
COLOR	Fija el color del fondo, el borde y el texto.

Sentencias y funciones relacionadas sólo con el modo TEXTO

WIDTH	Fija la anchura de la pantalla de texto.
LOCATE	Fija la posición del cursor en la pantalla de texto.
TAB	Fija la posición horizontal del cursor en la línea actual.
CSRLIN	Devuelve la posición vertical del cursor.
POS (0)	Devuelve la posición horizontal del cursor.

Sentencias y funciones relacionadas sólo con los modos GRAFICOS

CIRCLE	Dibuja un círculo.
DRAW	Dibuja de acuerdo al marco lenguaje gráfico.
LINE	Dibuja líneas, cuadrados y rectángulos.
PAINT	Pinta con el color de la tinta.
PSET	Pone un punto.
PRESET	Quita un punto.
POINT	Devuelve el color de un punto especificado.

Sentencias relacionadas con el procesador de video (VDP)

VPOKE	Ejecuta una sentencia POKE en la video RAM.
VPEEK	Ejecuta una sentencia PEEK en la video RAM.
VDP	Devuelve los valores de los registros del VDP.
BASE	Devuelve las direcciones de comienzo de las tablas de la RAM de video.

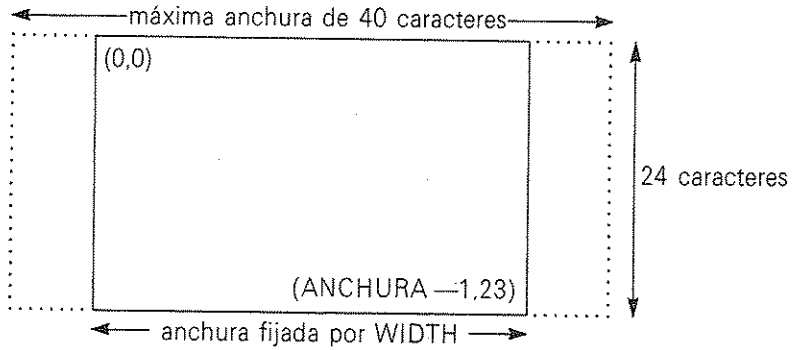
El MODO 0, con 40 caracteres por línea es, en los ordenadores MSX, el que mayor número de caracteres puede tener en una línea. En este modo puedes escribir y editar tu programa. Observarás que los programas en este modo son más fáciles de leer.

El MODO 0, sin embargo, tiene alguna desventaja. Por ejemplo, los caracteres se visualizan en un formato comprimido, o sea, 6 × 8 en vez de 8 × 8, y por ello algunos caracteres gráficos aparecen cortados. Los dos trazos de la derecha de un

carácter no se verán por completo. Sin embargo, esto no afecta a los caracteres alfanuméricos.

La anchura por defecto de la pantalla en MODO 0 es de 37 caracteres. Puedes incrementar la anchura a 40 caracteres utilizando la sentencia WIDTH. Las coordenadas de la esquina superior izquierda son (0,0), mientras que las de la esquina inferior derecha son (ANCHURA −1,23); en la anchura estándar son (38,23).

MODO 0: 40 × 24 MODO TEXTO **PANTALLA 0**



Puedes posicionar el cursor de texto mediante TAB y LOCATE. Para encontrar dónde se localiza el cursor emplea las funciones POS (0) y CSRLIN.

En MODO 0 no puedes utilizar *sprites*, y sólo dos de los dieciséis colores a la vez, aunque la combinación de esos dos colores está a tu elección. El color estándar es el mismo que en MODO 1: blanco para el texto y azul para el fondo. Observa que el MODO 0 no tiene borde. Simplemente desaparece de la pantalla; por tanto, no tiene sentido dar un color al borde en MODO 0. Advierte que el color de la pantalla cambiar inmediatamente después de la ejecución de la sentencia COLOR.

A diferencia del modo gráfico eres libre de utilizar sentencias INPUT, y las funciones de las teclas F1, ..., F10 se mostrarán en la línea 23, a menos que las borres mediante KEY OFF.

Todas las instrucciones y funciones gráficas en este modo serán tratadas como errores de llamada ilegal a una función *(Illegal function call)*. Ten cuidado con esto.

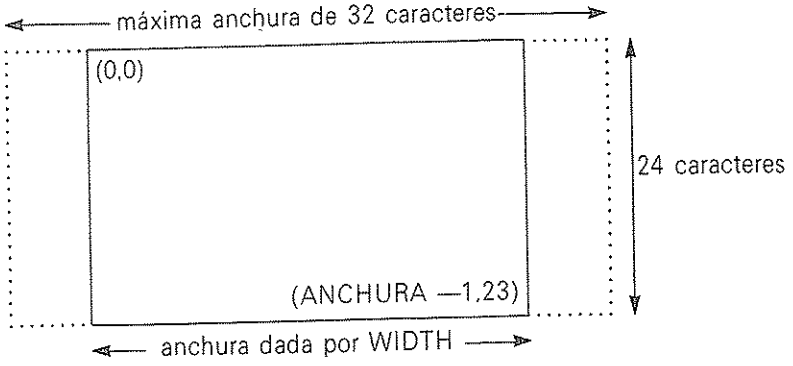
El MODO 1 tiene una resolución de 32 × 24 caracteres, pero no gráficos. En este modo puedes escribir y editar programas. Emplea caracteres de 8 × 8 sin ningún corte (como en el MODO 0).

La anchura estándar de la pantalla se puede incrementar a 32 caracteres mediante la instrucción WIDTH. La anchura estándar es más pequeña porque algunos televisores y monitores no pueden visualizar la pantalla completa.

Las coordenadas de la esquina superior izquierda son (0,0), mientras que las de la esquina inferior derecha son (ANCHURA −1,23) o (28,23) de un modo estándar.

MODO 1: 32 × 24 MODO TEXTO

PANTALLA 1



Puedes posicionar el cursor mediante TAB y LOCATE. Para saber las coordenadas del cursor emplea las funciones POS (0) y CRSLIN.

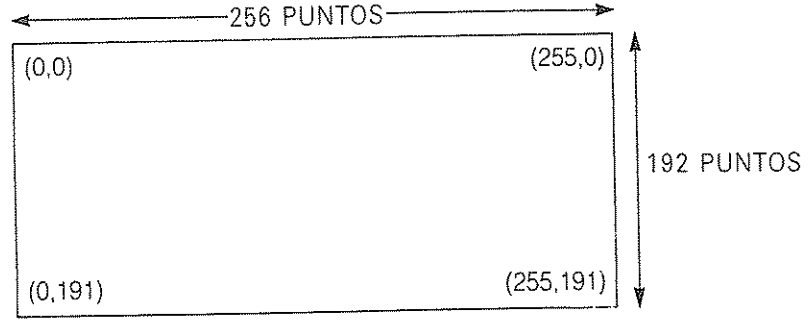
Sólo puedes utilizar dos de los dieciséis colores, aunque la combinación está enteramente a tu elección. Observa que el color de la pantalla cambiará inmediatamente después de haber ejecutado la orden COLOR.

A diferencia del modo gráfico, tienes plena libertad para emplear sentencias INPUT. Las funciones de las teclas F1 ,..., F10 se muestran en la línea 23, a menos que las borres mediante KEY OFF.

Todas las sentencias y funciones gráficas, excepto las de los *sprites*, se tratan como errores de llamada ilegal a una función *(Illegal function call)*.

MODO 2: 256 × 192 MODO DE ALTA RESOLUCION GRAFICA

PANTALLA 2



El MODO 2 es el modo gráfico más utilizado, proporcionándote una pantalla de alta resolución gráfica, con dieciséis colores. Puedes emplear *sprites* y el macrolenguaje gráfico de las sentencias DRAW. Es el modo que se usa en la mayoría de los programas de juego.

Su resolución horizontal es de 256 puntos, mientras que la vertical es de 192. La resolución de color, sin embargo, es diferente; siendo de 32 × 192. Esto implica que sólo puedes seleccionar dos colores por cada bloque horizontal de 8 puntos. Puedes seleccionar los colores de fondo y texto para cada bloque de 8 puntos, pero si trazas un punto con un tercer color, los ya trazados de dicho bloque cambiarán al color del punto recientemente pintado. Tus dibujos serán borrosos si no atiendes a la norma anterior. Si llevas cuidado con la resolución de colores de cada 8 puntos podrás realizar maravillas.

Más adelante te hablaremos sobre técnicas de dibujo avanzadas.

Puedes trabajar con *sprites* en este modo. Los *sprites* se sitúan en planos de pantalla diferentes de la pantalla principal. Con lo que puedes visualizar cualquier *sprite* sin que afecte a lo que haya en la pantalla principal. Observa que cuando un *sprite* pasa por una zona concreta de la pantalla, la pantalla principal permanece inmutable.

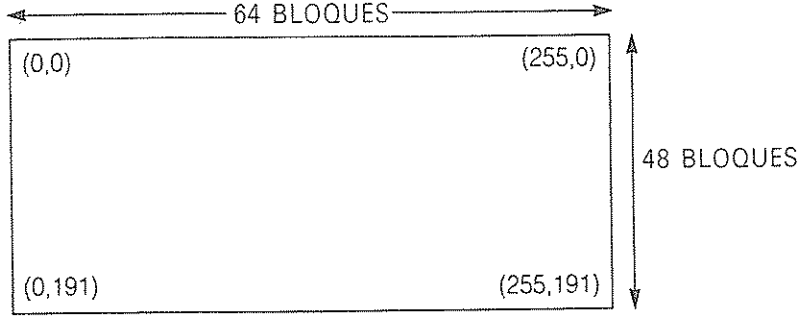
En el MODO 2 el contenido de las teclas de función no se visualiza. La sentencia PRINT no funciona en este modo. Para poder escribir en la pantalla gráfica, primero debes abrir un fichero para la pantalla y utilizar la instrucción PRINT. Te lo explicaremos con más detalle en la parte de escritura en modo gráfico.

La orden COLOR no cambia el color de la pantalla de forma inmediata. Para cambiar el color de toda la pantalla debes ejecutar primero el comando CLS.

No puedes utilizar instrucciones INPUT, ya que forzaría a la pantalla volver al modo de texto previamente utilizado.

MODO 3: 64 × 48 MODO MULTICOLOR DE BAJA RESOLUCION GRAFICA

PANTALLA 3



El MODO 3, multicolor, proporciona una baja resolución de 64 × 48 bloques individuales con su propio color (diferencia importante con el MODO 2). No obtendrás el efecto borroso como ocurría en el MODO 2, por encontrarse en baja resolución; hay pocas aplicaciones para este modo.

Sus coordenadas son las mismas que en el MODO 2, pero cadenas 4 × 4 puntos representan un bloque. Esto te permitirá dibujar y pintar con el mismo sistema que en el MODO 2.

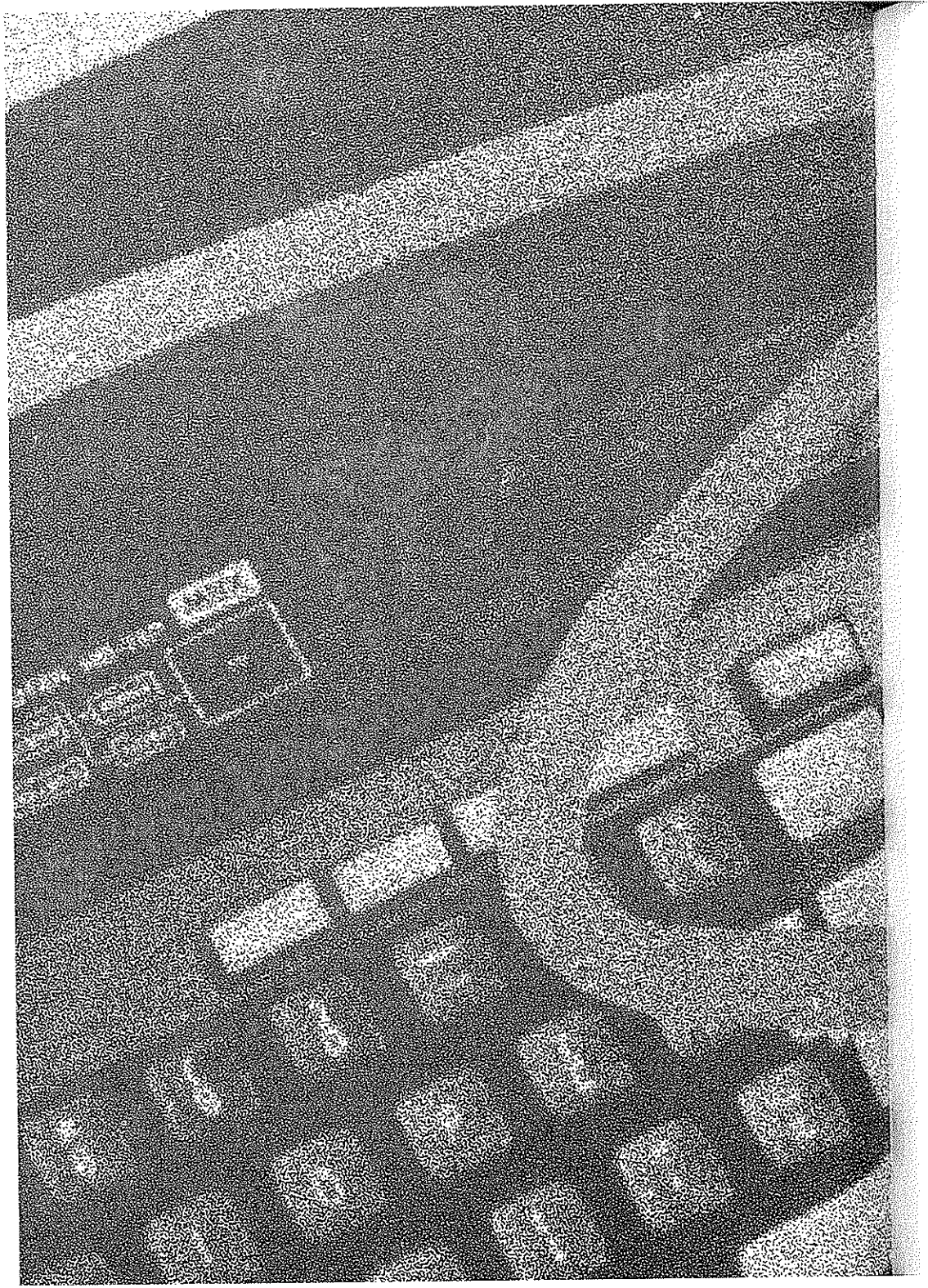
En este modo también puedes utilizar los *sprites* de igual forma que en el MODO 2.

No puedes visualizar el contenido de las teclas de función. La sentencia PRINT no funciona en este modo. Para poder escribir en este modo se debe abrir un fichero de pantalla y utilizar PRINT#.

El tamaño de los caracteres será cuatro veces mayor que en los modos 1 y 2. En la parte de escritura en los modos gráficos se explicará con más detalle.

La sentencia COLOR no cambia el color de la pantalla de forma inmediata. Para cambiar el color de toda la pantalla debes ejecutar previamente la sentencia CLS.

No puedes utilizar la instrucción INPUT en este modo, ya que forzaría que la pantalla volviera al modo de texto empleado anteriormente.



41

Gráficos avanzados (II)

El color en el MODO 2 de alta resolución

En esta sección se explica exclusivamente cómo se utilizan los colores en el MODO 2. La instrucción COLOR fija el color del texto, el fondo y el borde. Sin embargo, el color de fondo de toda la pantalla no cambiará a menos que previamente la limpies con CLS. Después de ejecutar la sentencia COLOR, cualquier punto, línea o figura dibujada estarán en el nuevo color del texto, a menos que especifiques otro color en las instrucciones de dibujo.

La resolución del color en el MODO 2 es de 32 × 192. Esto significa que sólo puedes seleccionar dos colores por cada bloque horizontal de 8 puntos. Puedes especificar un color de texto y otro de fondo para cada bloque de 8 puntos, pero si intentas trazar un punto con un tercer color en un bloque que ya tenía dos colores, los puntos con el color previo del texto de este bloque cambiarán al nuevo color del texto automáticamente. Si no llevas cuidado, puedes producir dibujos borrosos. Este pequeño programa te demostrará lo expuesto:

```
10 SCREEN 2
20 COLOR 4,15,15
30 CLS
40 LINE (0,0)-(4,191),4,BF
50 IF NOT STRIG(0) THEN 50
60 LINE (6,0)-(6,191),10
70 GOTO 70
```

LINEA 10 SELECCIONA EL MODO 2.
LINEA 20 FIJA EL FONDO EN BLANCO.
LINEA 30 LIMPIA LA PANTALLA PONIENDOLA BLANCA.
LINEA 40 DIBUJA UN RECTANGULO EN AZUL.
LINEA 50 ESPERA HASTA QUE PULSES LA BARRA ESPACIADORA.
LINEA 60 DIBUJA UNA LINEA EN AMARILLO PROXIMA AL RECTANGULO AZUL.
LINEA 70 MANTIENE LA PANTALLA GRAFICA.

Ejecuta este programa. Verás un rectángulo azul sobre el fondo blanco, en los límites de la pantalla; si ahora pulsas la barra espaciadora verás cómo se dibuja una línea amarilla, y el rectángulo también se volverá amarillo. El programa no traza ningún rectángulo amarillo sobre el azul. ¿Por qué cambia el color del rectángulo al trazar una línea amarilla cerca de él?

Para entenderlo, vamos a ver cómo se almacenan los atributos de color en la RAM de video.

Si tomásemos la parte superior izquierda de la pantalla y la aumentásemos, tendríamos una representación como la siguiente tabla (A para Azul y B para blanco):

Y X =	0	1	2	3	4	5	6	7	8	9	.	.	.
0	A	A	A	A	B	B	B	B	B	B			
1	A	A	A	A	B	B	B	B	B	B			
2	A	A	A	A	B	B	B	B	B	B			
3	A	A	A	A	B	B	B	B	B	B			
4													

Dentro de la RAM de video del MSX los colores se almacenan en binario, tal como tienes abajo. Cada bit se corresponde con un punto en la pantalla, 1 = color del texto; 0 = color del fondo. En nuestro caso el 1 representa el azul y el 0 el blanco:

	0	1	2	3	4	5	6	7	8	9	.	.	.
0	1	1	1	1	0	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	0	0	0	0	0	0	
2	1	1	1	1	0	0	0	0	0	0	0	0	
3	1	1	1	1	0	0	0	0	0	0	0	0	
4													

Sin embargo, esta tabla de memoria no revela el color asociado a cada punto. Existe otra tabla que nos indica la distribución de colores en la RAM de video, cuyo contenido son los colores del texto y el fondo para cada bloque de 8 × 1 puntos. Tomando la esquina superior izquierda de dicha tabla de diseños vemos cómo se almacenan en memoria el formato de diseño y el color.

TABLA DE DISEÑOS

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

TABLA DE ATRIBUTOS DE COLOR

1	0
4	15

Tal como ves, la tabla de diseños nos indica si un punto está al color del fondo o al del texto, y la de atributos de color nos proporciona los colores de fondo y del texto de cada bloque de 8 puntos, por cuanto es físicamente imposible pintar con más de tres colores en un bloque. Por tanto, cuando intentas dibujar una línea amarilla en la sexta columna lo primero que se hace es cambiar el color del texto en la tabla de atributos, con el código de color amarillo, o sea, 10, y entonces el punto 6 se fija con este nuevo color del texto:

TABLA DE DISEÑOS

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

TABLA DE ATRIBUTOS DE COLOR

1	0
10	15

Los puntos 0, 1, 2 y 3 permanecen con el color del texto, pero éste cambia de azul a amarillo y, por tanto, el color de los puntos 0, 1, 2 y 3 también cambiará.

La explicación es sencilla cuando ves estas tablas de memoria, ¿verdad?

¿Qué hacemos ahora con este problema? Sólo hay una respuesta: intenta no poner más de tres colores en cada bloque de 8 puntos.

Por ejemplo, en el programa anteriormente expuesto puedes cambiar la línea amarilla para que se trace en distinto bloque que el rectángulo azul, moviendo la línea dos puntos a la izquierda. El programa quedará así:

```
10 SCREEN 2
20 COLOR 4,15,15
30 CLS
40 LINE (2,0)-(6,191),4,BF
50 IF NOT STRIG(0) THEN 50
60 LINE (8,0)-(8,191),10
70 GOTO 70
```

LINEA 10 SELECCIONA EL MODO 2.
LINEA 20 PONE EL FONDO EN BLANCO.
LINEA 30 LIMPIA LA PANTALLA PONIENDOLA BLANCA.
LINEA 40 DIBUJA UN RECTANGULO AZUL.
LINEA 50 ESPERA HASTA QUE PULSES LA BARRA ESPACIADORA.
LINEA 60 DIBUJA UNA LINEA AMARILLA AL LADO DEL RECTANGULO AZUL.
LINEA 70 MANTIENE LA PANTALLA GRAFICA.

Advierte los cambios que se han realizado en los especificadores de coordenadas de las líneas 40 y 60.

Nota: Si deseas obtener más detalles del modo de almacenar la pantalla en la RAM de video consulta la parte de la RAM de video.

42
Gráficos avanzados (III)

Cómo escribir en las pantallas gráficas empleando ficheros

La escritura en los modos gráficos no es tan directa como en los modos de texto. Has de abrir un fichero para la pantalla gráfica y escribir mediante la instrucción PRINT#. De hecho, puedes abrir un fichero para una pantalla de texto si lo deseas, pero resulta redundante.

Este programa de demostración escribirá en los cuatro modos de pantalla, utilizando ficheros. La línea 30 abre un fichero de acuerdo con el modo (M) de pantalla seleccionado. Ejecuta OPEN"CRT:" AS #1 cuando M es menor que 2. O sea, abre un fichero para una pantalla de texto "CRT:" con un número de fichero, #1. Cuando M es mayor o igual que 2, entonces se ejecuta OPEN"GRP:"AS #1, que abre un fichero para las pantallas gráficas "GRP:" con un número de fichero, #1. Todo fichero abierto debe cerrarse una vez que acabes de escribir. En la línea 60, CLOSE#1 cierra el correspondiente fichero.

```
10 FOR M=0 TO 3
20 SCREEN M
30 IF M<2 THEN OPEN "CRT:" AS #1 ELSE OPEN "GRP:"
AS #1
40 PRINT#1,"ESTAS EN EL MODO";M
50 FOR G=1 TO 1000: NEXT G
60 CLOSE#1
70 NEXT M
```

LINEA 10 BUCLE DE 0 A 3.
LINEA 20 SELECCIONA EL MODO DE PANTALLA.
LINEA 30 SI ES UN MODO TEXTO, ENTONCES ABRIR "CRT:"; EN CASO CONTRARIO, ABRIR "GRP:" PARA PANTALLAS GRAFICAS.
LINEA 40 ESCRIBE UN MENSAJE.
LINEA 50 RETARDO.
LINEA 60 CIERRA EL FICHERO.
LINEA 70 SIGUIENTE MODO.

Verás cómo la pantalla va cambiando de modo, mientras se escribe el mensaje "ESTAS EN EL MODO...".

Observa que en la pantalla de alta resolución se escribe con el mismo formato que en el modo de texto 1; o sea, 32 × 24 caracteres. En el MODO 3 multicolor el texto tiene un tamaño cuatro veces mayor que en el modo de texto normal. Esto es debido a que el modo multicolor es de baja resolución, y cuando escribe caracteres lo hace en bloques gráficos cuatro veces mayores que su tamaño normal.

Cómo escribir en el MODO 2 de alta resolución gráfica

Cuando escribas en el MODO 2 gráfico no puedes emplear la sentencia LOCATE para situar los caracteres. Esto se debe a que el ordenador no utiliza el cursor de texto en el modo gráfico y, por tanto, el ordenador escribe a partir de la última posición del cursor gráfico. Para escribir a partir de una determinada posición emplea la instrucción DRAW y mueve el cursor gráfico con el macrocomando gráfico BM (mover sin marcar el recorrido).

Programa ejemplo: PRINT# y PRINT# USING en el MODO 2 gráfico.

```
10 SCREEN 2
20 OPEN "GRP:" AS #1
30 DRAW "BM100,100"
40 PRINT#1,"SOCORRO"
50 DRAW "BM102,102"
60 PRINT#1,"SOCORRO"
70 DRAW "BM100,120"
80 PRINT#1,USING"###.######";ATN(1)*4
90 GOTO 90
```

LINEA 10 SELECCIONADO EL MODO 2 PARA LA PANTALLA.
LINEA 20 ABRE UN FICHERO GRP.
LINEA 30 POSICIONA EL CURSOR GRAFICO Y ESCRIBE.
LINEA 50 POSICIONA EL CURSOR.
LINEA 60 ESCRIBE OTRA VEZ SOCORRO.

LINEA 70 POSICIONA DE NUEVO EL CURSOR.
LINEA 80 PRINT# USING.
LINEA 90 MANTIENE EL MODO GRAFICO.

En el ejemplo anterior habrás visto dos mensajes "SOCORRO" superpuestos. Es debido a que en el modo gráfico el ordenador no borra antes de escribir. Así puedes crear algunos efectos especiales, pero normalmente es un inconveniente. El único modo de borrar lo escrito es mediante la sentencia LINE, superponiendo un rectángulo del color de fondo sobre la parte a borrar.

Si incluyes la línea:

```
45 LINE(100,100)-STEP(56,8),4,BF
```

se borrará el mensaje "SOCORRO" de la línea 40.

Como ya viste, también puedes utilizar PRINT USING en la forma de PRINT# USING. Todas las variedades sintácticas de PRINT USING son iguales para los modos gráficos y para los modos de texto.

Observa que ATN(1)*4 = PI = 3,14159...

Cómo escribir mediante colores en el MODO 2 gráfico

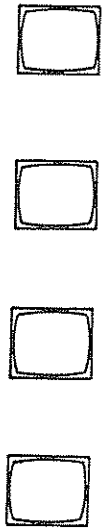
Una de las ventajas de la escritura en el modo gráfico es que puedes utilizar los dieciséis colores. Para escribir con un color específico todo lo que debes hacer es cambiar el color del texto antes de ejecutar PRINT.

Aquí tienes un ejemplo que ilustra la escritura con los dieciséis colores:

```
10 COLOR 1,15,15
20 SCREEN 2
30 OPEN "GRP:" AS #1
40 FOR I=0 TO 15
50 COLOR I
60 DRAW "BM+8,0"
70 PRINT#1,"CODIGO DE COLOR";I
80 NEXT I
90 GOTO 90
```

LINEA 10 EL COLOR DEL FONDO ES EL BLANCO.
LINEA 20 PANTALLA EN MODO 2.
LINEA 30 ABRIR UN FICHERO GRP CON NUMERO #1.
LINEA 40 BUCLE.
LINEA 50 COLOR DE BUCLE.
LINEA 60 MUEVE EL CURSOR GRAFICO.
LINEA 70 MENSAJE.
LINEA 80 SIGUIENTE BUCLE.
LINEA 90 MANTIENE LA PANTALLA.

El resultado del programa anterior es:



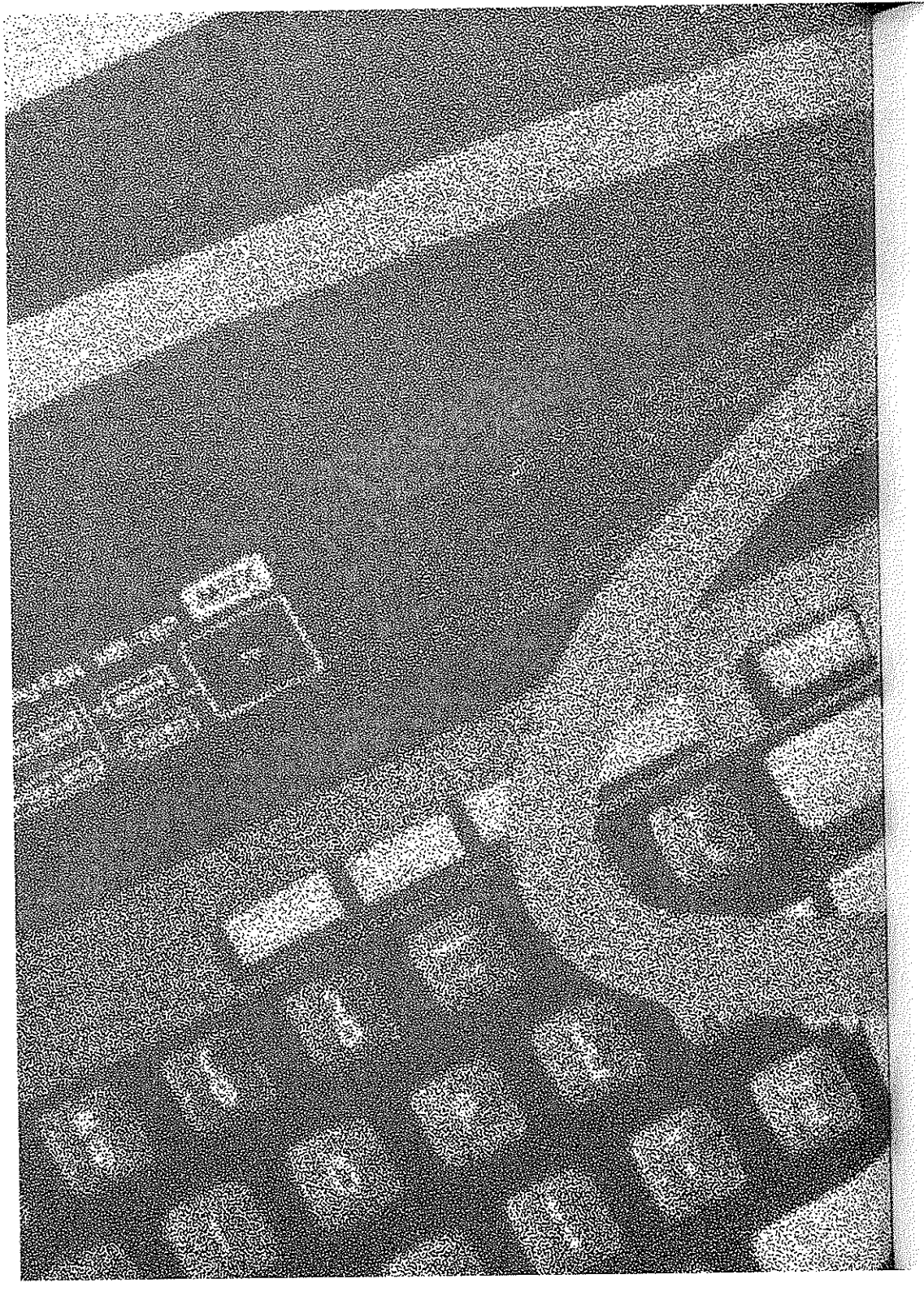
```
CODIGO DE COLOR 0     TRANSPARENTE
CODIGO DE COLOR 1     NEGRO
CODIGO DE COLOR 2     VERDE (MEDIO)
CODIGO DE COLOR 3     VERDE (CLARO)
CODIGO DE COLOR 4     AZUL (OSCURO)
CODIGO DE COLOR 5     AZUL (CLARO)
CODIGO DE COLOR 6     ROJO (OSCURO)
CODIGO DE COLOR 7     CIAN
CODIGO DE COLOR 8     ROJO (MEDIO)
CODIGO DE COLOR 9     ROJO (CLARO)
CODIGO DE COLOR 10    AMARILLO (OSCURO)
CODIGO DE COLOR 11    AMARILLO (CLARO)
CODIGO DE COLOR 12    VERDE (OSCURO)
CODIGO DE COLOR 13    MAGENTA
CODIGO DE COLOR 14    GRIS
CODIGO DE COLOR 15    BLANCO
```

Cómo imprimir en el MODO gráfico multicolor 3

En el MODO 3 el texto tiene un tamaño cuatro veces mayor que en un modo normal, por tener la pantalla una baja resolución. Para escribir caracteres se emplean bloques gráficos, resultando enormes. Este programa te lo demostrará.

```
10 SCREEN 3
20 OPEN "GRP:" AS #1
30 DRAW "BM0,0"
40 PRINT#1,"PI"
50 DRAW "BM0,65"
60 PRINT#1,USING "#.######";ATN(1)*4
70 GOTO 70
```

LINEA 10 SELECCIONA EL MODO MULTICOLOR.
LINEA 20 ABRE UN FICHERO PARA LA PANTALLA.
LINEA 30 POSICIONA EL CURSOR.
LINEA 40 ESCRIBE PI
LINEA 50 REPOSICIONA EL CURSOR.
LINEA 60 ESCRIBE EL VALOR DE PI.
LINEA 70 MANTIENE LA PANTALLA GRAFICA.



43

Gráficos avanzados (IV)

Los *sprites*

El procesador de video del MSX tiene la capacidad de generar *sprites*. Los *sprites* son caracteres o figuras definidas por el usuario, que pueden visualizarse en pantalla sin afectar a la pantalla de fondo, ya que se sitúan en diferentes planos de pantalla. Puedes hacer que se escondan uno detrás de otro y también moverlos. Te brindan la posibilidad de hacer juegos de un modo sencillo sin preocuparte por la pantalla de fondo.

Tamaño de *sprites*

Puedes emplear cuatro tamaños de *sprites*, pero en un instante determinado sólo podrás utilizar un único tamaño. El tamaño se determina mediante la instrucción SCREEN.

	TAMAÑO	
SCREEN,0	8 × 8	punto normal
SCREEN,1	8 × 8	punto ampliado a 16 × 16
SCREEN,2	16 × 16	punto normal
SCREEN,3	16 × 16	punto ampliado a 32 × 32

La ampliación te da una resolución de 2 × 2 puntos.

Definición de *sprites*: SPRITE$

Puedes definir tus *sprites* empleando la instrucción SPRITE$. Puedes tener hasta 256 modelos de *sprites* cuando trabajes con los tamaños de *sprite* 0 o 1 (8 × 8 puntos), o 64 con el tamaño de *sprite* 2 o 3 (16 × 16 puntos).

Sintaxis

SPRITE$(<entero>) = <cadena de caracteres>.
<entero> = número de modelos *sprite*.
<entero> = de 0 a 255 cuando el tamaño del *sprite* es 0 o 1.
<entero> = de 0 a 63 cuando el tamaño del *sprite* es 2 o 3.

SPRITE$ es una cadena de caracteres. La longitud de la cadena del *sprite* es de 32 bytes (o caracteres), pero para *sprites* pequeños de 8 × 8 sólo necesitas definir los ocho primeros bytes.

Cada byte de la cadena del *sprite* representa una fila de 8 puntos y se suma a la cadena del *sprite* el carácter con código ASCII igual al del byte. Puedes hacerlo de un modo sencillo mediante la función CHR$(). Por ejemplo, para formar un *sprite* de 8 × 8:

SPRITE$(<entero>) = CHR$(<1.ª línea>) + CHR$(2.ª línea>) + CHR$(<3.ª línea>) + CHR$(<4.ª línea>) + CHR$(<5.ª línea>) + CHR$(<6.ª línea>) + CHR$(<7.ª línea>) + CHR$(<8.ª línea>).

La función CHR$() se puede reemplazar por el carácter, si sabes cuál es. Para definir un *sprite* de 16 × 16 tienes que definir los 33 bytes que lo componen.

Aquí tienes un método para definir *sprites* de 8 × 8.

Primero dibuja tu *sprite* en un papel.

128	64	32	16	8	4	2	1		BINARIO		DECIMAL
·	·	·	1	·	·	·	·	=	&B00010000	=	16
·	·	1	1	·	·	·	·	=	&B00110000	=	48
·	1	1	1	·	·	·	·	=	&B01110000	=	112
1	1	1	1	1	1	1	1	=	&B11111111	=	255
1	1	1	1	1	1	1	1	=	&B11111111	=	255
·	1	1	1	·	·	·	·	=	&B01110000	=	112
·	·	1	1	·	·	·	·	=	&B00110000	=	48
·	·	·	1	·	·	·	·	=	&B00010000	=	16

Forma una expresión de caracteres como la siguiente:

SPRITE$(0) = CHR$(16) + CHR$(48) + CHR$(112) + CHR$(255) + CHR$(255) + CHR$(112) + CHR$(48) + CHR$(16).

Ejemplo:

La manera más sencilla de definir *sprites* es mediante los números binarios; te resultará más fácil.

```
10  SCREEN 2,0
20  FOR I=1 TO 8
30  READ A$
40  S$=S$+CHR$(VAL("&B"+A$))
50  NEXT I
60  SPRITE$(0)=S$
70  PUT SPRITE 0,(100,100),15,0
80  GOTO 80
90  REM Datos en binario
100 DATA 00010000
110 DATA 00110000
120 DATA 01110000
130 DATA 11111111
140 DATA 11111111
150 DATA 01110000
160 DATA 00110000
170 DATA 00010000
```

LINEA 10 PANTALLA DE ALTA RESOLUCION, TAMAÑO NORMAL DE SPRITE.
LINEA 30 LEE EL NUMERO BINARIO COMO CADENA DE CARACTERES.
LINEA 40 S$ ES UNA CADENA DE CARACTERES TEMPORAL.
LINEA 60 DEFINE EL SPRITE 0.
LINEA 70 PONE EL SPRITE 0 EN X = 100, Y = 100, COLOR BLANCO, Y PLANO DE SPRITE 0.

Resultado: verás una flecha en mitad de la pantalla.

Más adelante veremos con más detalle la instrucción PUT SPRITE.

Si sabes calular los valores decimales de cada byte, entonces se puede escribir el programa en una versión más corta. El programa de arriba se puede reducir a las siguientes líneas:

```
10 SCREEN 2,0
20 SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255
)+CHR$(255)+CHR$(112)+CHR$(48)+CHR$(16)
70 PUT SPRITE 0,(100,100),15,0
80 GOTO 80
```

LINEA 10 PANTALLA DE ALTA RESOLUCION.
LINEA 20 DEFINICION DEL SPRITE.
LINEA 70 PONE EL SPRITE 0 EN X = 100, Y = 100, COLOR BLANCO, PLANO DE SPRITE 0.

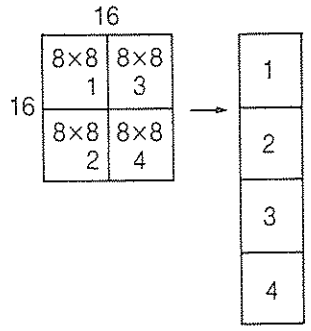
Cómo definir un *sprite* de 16 × 16

128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
.	.	.	1	1	1	1	1	1	1	1	1	1	.	.	.
.	.	1	1	1	1	1	1	1	1	1	1	1	1	.	.
.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	.
1	1	.	.	.	1	1	1	1	1	1	.	.	.	1	1
1	1	.	.	.	1	1	1	1	1	1	.	.	.	1	1
1	1	.	.	1	.	.	.	.	.	.	1	.	.	1	1
.	1	1	.	1	.	.	.	.	.	.	1	.	1	1	.
.	1	1	.	.	1	.	.	.	.	1	.	.	1	1	.
.	.	1	1	.	.	1	1	1	1	.	.	1	1	.	.
.	.	1	1	.	1	.	.	.	.	1	.	1	1	.	.
.	.	.	1	1	.	1	1	1	1	.	1	1	.	.	.
.	.	.	1	1	.	.	.	.	.	.	1	1	.	.	.
.	.	.	.	1	1	.	.	.	.	1	1	.	.	.	.
.	.	.	.	1	1	.	.	.	.	1	1	.	.	.	.
.	.	.	.	.	1	1	.	.	1	1	.	.	.	.	.
.	.	.	.	.	.	1	1	1	1	.	.	.	.	.	.
.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.

El diagrama arriba indicado, expresado en binario y decimal:

00011111	= 31	11111000	= 248
00111111	= 63	11111100	= 252
01100000	= 96	00000110	= 6
11000111	= 199	11100011	= 227
11001000	= 200	00010011	= 19
01101000	= 104	00010110	= 22
01100100	= 100	00100110	= 38
00110011	= 51	11001100	= 204
00110100	= 52	00101100	= 44
00011011	= 27	11011000	= 216
00011000	= 24	00011000	= 24
00001100	= 12	00110000	= 48
00001100	= 12	00110000	= 48
00000110	= 6	01100000	= 96
00000011	= 3	11000000	= 192
00000001	= 1	10000000	= 128

Los modelos de los *sprites* de 16 × 16 se almacenan en la RAM de video de la siguiente forma:



Un SPRITE$ de 16 × 16 = "*sprite* del cuadrante 1" + "*sprite* del cuadrante 2" + "*sprite* del cuadrante 3" + "*sprite* del cuadrante 4".

Aquí tienes un programa que define y visualiza el *sprite* de 16 × 16 que hemos diseñado anteriormente. Los datos se almacenan en las instrucciones DATA y suman uno a uno a SPRITE$(0) dentro del bucle FOR/TO/NEXT.

```
10  SCREEN 2,2
20  FOR I=1 TO 32
30  READ B%
40  S$=S$+CHR$(B%)
50  NEXT I
60  SPRITE$(0)=S$
70  PUT SPRITE 0,(100,100),15,0
80  GOTO 80
90  DATA 31,63,96,199,200,104,100,51
100 DATA 52,27,24,12,12,6,3,1
110 DATA 248,252,6,227,19,22,38,204
120 DATA 44,216,24,48,48,96,192,128
```

LINEA 10 SELECCIONA LA PANTALLA EN MODO 2 CON UN TAMAÑO PARA EL SPRITE DE 16 × 16.
LINEA 20 BUCLE PARA LEER 32 BYTES.
LINEA 30 LEE.
LINEA 40 AÑADE DATOS A S$.
LINEA 60 DEFINE LA CADENA DEL SPRITE 0.
LINEA 70 PONE EL SPRITE 0 EN EL PLANO 0 DANDOLE COLOR BLANCO.
LINEA 90 DATOS PARA EL CUADRANTE 1.
LINEA 100 DATOS PARA EL CUADRANTE 2.
LINEA 105 DATOS PARA EL CUADRANTE 3.
LINEA 110 DATOS PARA EL CUADRANTE 4.

Cómo situar un *sprite*: en pantalla: PUT SPRITE

Delante de los planos de texto/gráficos hay 32 planos de *sprites*, siendo el plano *sprite* número 0 el más exterior de todos. O sea, cuanto menor sea el número de plano, mayor es su prioridad. Si hay dos *sprites* superpuestos, el que tenga el número de plano de *sprite* menor, se verá por delante, quedando el otro escondido detrás de él.

Puedes definir hasta 256 modelos de *sprite* diferentes, pero sólo puedes tener un *sprite* en cada plano. Esto limita el número máximo de *sprites* en pantalla a 32.

En una línea horizontal sólo puedes tener cuatro *sprites* como máximo.

Para situar un *sprite* en pantalla usa la sentencia PUT SPRITE. Esta tiene la siguiente sintaxis:

PUT SPRITE <número de plano *sprite*>[,<especificación de coordenadas>],[<color>][,<número de modelo>]
<número del plano *sprite*> tiene un rango de 0 a 31
<especificación de coordenadas> indica el punto donde se situará la esquina superior izquierda del *sprite* en pantalla.

Hay dos formatos para <especificación de coordenadas>:

1. (<X-coordenada>,<Y-coordenada>)
   (especifica un punto absoluto).
2. STEP (<X-incremento>,<Y-incremento>)
   (de las coordenadas del punto relativas al último punto que se hizo referencia).

<X-incremento>, <Y-incremento>, <X-coordenada>, <Y-coordenada> pueden ser variables o expresiones, así como simples constantes numéricas. Lo que significa que los *sprites* se pueden controlar mediante variables, permitiendo el movimiento por pantalla.

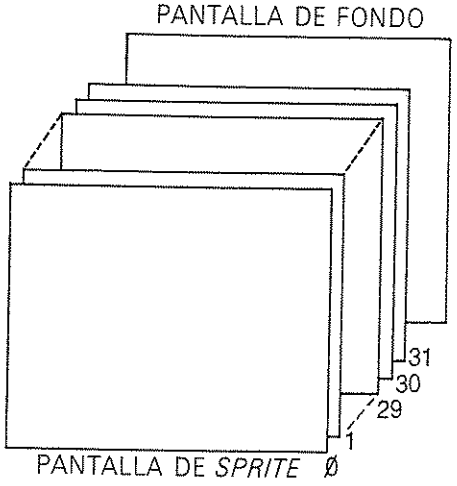
Si <especificación de coordenada> se omite, la instrucción PUT SPRITE situará el *sprite* en el último punto al que se hizo referencia.

Ejemplos:

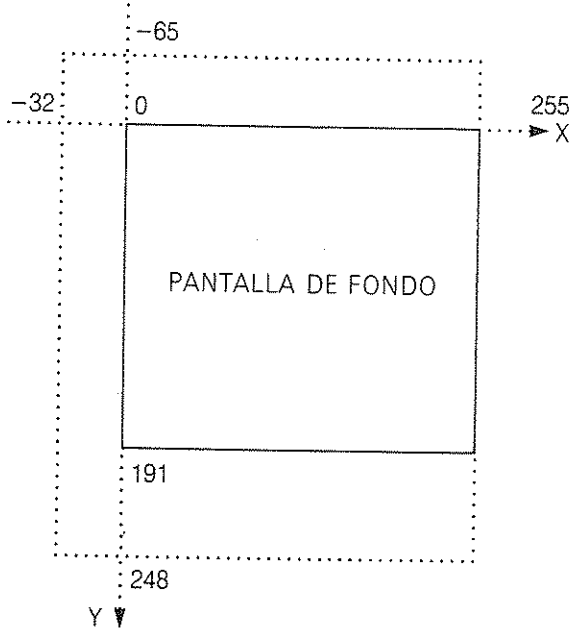
```
PUT SPRITE 0,(50,50),1,1
PUT SPRITE 1,STEP(10,10),12,2
```

La pantalla de *sprites* es ligeramente más grande que la principal. La coordenada X está entre −32 y 255, y la coordenada Y entre −65 y 248. La pantalla principal tiene un rango menor, por lo que si el *sprite* se sitúa fuera de las coordenadas de esta pantalla, éste quedará escondido total o parcialmente.

Los *sprites* tienen la propiedad de tener la pantalla unida por sus extremos. Por lo que si un *sprite*, en su movimiento, se sale fuera de la pantalla, aparece de nuevo por el extremo opuesto.



La pantalla de *sprite:*



Movimiento de *sprites*:

Variando el valor del especificador de coordenadas de la instrucción PUT PRITE, puedes mover los *sprites* por toda la pantalla. Cuando se ejecuta la instrucción PUT SPRITE se borra automáticamente el *sprite* que se hallaba en dicha pantalla de *sprites*.

Este programa te muestra un cuadrado que se mueve de derecha a izquierda, desapareciendo por la izquierda y reapareciendo por la derecha:

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 FOR X=200 TO -200 STEP -1
40 PUT SPRITE 0,(X,100),1,0
50 FOR D=1 TO 50: NEXT
60 NEXT
70 END
```

LINEA 10 SELECCIONA PANTALLA, TAMAÑO NORMAL.
LINEA 20 DEFINE UN SPRITE CUADRADO.
LINEA 40 PONE EL SPRITE EN X,100.
LINEA 50 RETARDO.

El color de los *sprites*

Puedes usar los dieciséis colores. Sin embargo, sólo puedes utilizar un color por cada *sprite*; es imposible tener un *sprite* multicolor. Para los *sprites* multicolor has de situar dos o más *sprites* de diferentes colores, uno encima de otro, en diferentes planos de *sprites* y moverlos conjuntamente.

El color de fondo de un *sprite* es siempre transparente y, por tanto, puedes ver a través de él.

Este ejemplo define dos *sprites* 0 y 1. El *sprite* 0, en blanco, y el *sprite* 1, en amarillo oscuro. En la pantalla verás a los dos *sprites* separados, y también superpuestos aparentando un *sprite* multicolor.

```
10 SCREEN 2,0
20 SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255
)+CHR$(255)+CHR$(112)+CHR$(48)+CHR$(16)
30 SPRITE$(1)=CHR$(224)+CHR$(192)+CHR$(128)+CHR$(0
)+CHR$(0)+CHR$(128)+CHR$(192)+CHR$(224)
40 PUT SPRITE 0,(20,20),15,0
50 PUT SPRITE 1,(40,40),10,1
60 PUT SPRITE 2,(60,60),15,0
70 PUT SPRITE 3,(60,60),10,1
80 GOTO 80
```


LINEA 10 SELECCIONA LA PANTALLA, TAMAÑO NORMAL.
LINEA 20 DEFINE EL SPRITE 0.
LINEA 30 DEFINE EL SPRITE 1.
LINEA 40 PONE EL SPRITE 0 CON COLOR BLANCO.
LINEA 50 PONE EL SPRITE 1 CON COLOR AMARILLO OSCURO.
LINEA 60 PONE LOS SPRITES 2 y 3 EN LAS MISMAS COORDENADAS UTILIZANDO DOS COLORES, EN DIFERENTES PLANOS DE SPRITE.

Escondiendo *sprites*

Asignando a la coordenada Y un número especial, puedes esconder tus *sprites:*

Y = 208

Si a la coordenada Y la asignas el valor 208 (&HD0), desaparecerán todos los planos de *sprite* con mayor número hasta que cambies el valor de Y.

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 PUT SPRITE 0,(20,20),15,0
40 PUT SPRITE 1,(40,40),15,0
50 PUT SPRITE 2,(60,208),15,0
60 PUT SPRITE 3,(80,80),15,0
70 PUT SPRITE 4,(100,100),15,0
80 GOTO 80
```

LINEA 10 SELECCIONA LA PANTALLA, TAMAÑO NORMAL.
LINEA 20 DEFINE UN SPRITE CUADRADO.
LINEA 50 Y = 208. INUTILIZA LOS PLANOS DE SPRITE CON UN NUMERO MAYOR DE 2: LOS SPRITES DE LAS LINEAS 60 Y 70 NO APARECEN.

Si a la coordenada Y le damos el valor 209 (&HD1) el *sprite* desaparece de pantalla.

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 PUT SPRITE 0,(20,20),15,0
40 PUT SPRITE 1,(40,40),15,0
50 PUT SPRITE 2,(60,209),15,0
60 PUT SPRITE 3,(80,80),15,0
70 PUT SPRITE 4,(100,100),15,0
80 GOTO 80
```

LINEA 10 SELECCIONA PANTALLA, TAMAÑO NORMAL.
LINEA 20 DEFINE UN SPRITE CUADRADO.
LINEA 30 SPRITE CUADRADO EN PLANO 0.
LINEA 40 SPRITE CUADRADO EN PLANO 1.
LINEA 50 Y = 209. DESAPARECE EL SPRITE.
LINEA 60 SPRITE CUADRADO EN PLANO 3.
LINEA 70 SPRITE CUADRADO EN PLANO 4.

La "quinta" regla de los *sprites*

El número máximo de *sprites* que puede haber en una línea horizontal es cuatro. Si violas esta regla sólo se verán los cuatro últimos *sprites* referidos; el resto no se mostrará en dicha línea. En el registro de estado del VDP tienes el número del "quinto" *sprite* que rompió la regla; lo puedes obtener mediante la función VDP. (Véase la parte correspondiente a la función VDP.)

Ejemplo:

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 PUT SPRITE 0,(20,100),15,0
40 PUT SPRITE 1,(40,100),15,0
50 PUT SPRITE 2,(60,100),15,0
60 PUT SPRITE 3,(80,100),15,0
70 PUT SPRITE 4,(100,104),15,0
80 GOTO 80
```

LINEA 10 SELECCIONA PANTALLA, TAMAÑO NORMAL.
LINEA 20 DEFINE UN SPRITE CUADRADO.
LINEA 30 SPRITE CUADRADO EN LINEA 100.
LINEA 40 SPRITE CUADRADO EN LINEA 100.
LINEA 50 SPRITE CUADRADO EN LINEA 100.
LINEA 60 SPRITE CUADRADO EN LINEA 100.
LINEA 70 SPRITE CUADRADO EN LINEA 104.

En el ejemplo de arriba verás cuatro *sprites* cuadrados, en la misma línea horizontal (100). Del quinto *sprite*, que está en la línea 104, se ve la mitad; la otra mitad está escondida, a consecuencia de la regla del "quinto" *sprite*. Si estuviera en la línea 108 se vería por entero.

Animación de *sprites*

Puedes programar con tu MSX una simple animación de *sprites*. Todo lo que tienes que hacer es intercambiar dos *sprites* rápidamente, con lo que parecerá que están vivos.

El siguiente programa te muestra un invasor del espacio al que se le mueven las piernas. El SPRITE$(0) es el invasor con las piernas cerradas, y el SPRITE$(1) es el invasor con las piernas abiertas.

```
10 SCREEN 2,1
20 SPRITE$(0)=CHR$(60)+CHR$(126)+CHR$(129)+CHR$(21
9)+CHR$(126)+CHR$(36)+CHR$(36)+CHR$(36)
30 SPRITE$(1)=CHR$(60)+CHR$(126)+CHR$(129)+CHR$(21
9)+CHR$(126)+CHR$(36)+CHR$(60)+CHR$(129)
40 PUT SPRITE 0,(100,100),11,0
50 FOR I=1 TO 500: NEXT
60 PUT SPRITE 0,(100,100),11,1
70 FOR I=1 TO 500: NEXT
80 GOTO 40
```

LINEA 20 INVASOR CON PIERNAS CERRADAS.
LINEA 30 INVASOR CON PIERNAS ABIERTAS.
LINEA 40 SPRITE 0 (PIERNAS CERRADAS) AMARILLO.
LINEA 50 RETARDO.
LINEA 60 SPRITE 1 (PIERNAS AIBERTAS) AMARILLO.
LINEA 70 RETARDO.

Programa de demostración

Con este ejemplo verás dos planetas girando alrededor de una estrella. El sistema aparecerá por el lado superior izquierdo de la pantalla.

```
10  SCREEN 2,0
20  COLOR 15,1,1
30  CLS
40  SPRITE$(0)=CHR$(126)+STRING$(6,CHR$(255))+CHR$
(126)
50  SPRITE$(1)=STRING$(3,CHR$(0))+CHR$(24)+CHR$(24
)+STRING$(3,CHR$(0))
60  FOR I=0 TO 6.28 STEP .2
70  X=X+1.5
80  Y=Y+1
90  X1=30*COS(I)
100 Y1=30*SIN(I)
110 X2=15*COS(I)
120 Y2=15*SIN(I)
130 PUT SPRITE 0,(X,Y),11,0
140 PUT SPRITE 1,(X1+X,Y1+Y),9,1
150 PUT SPRITE 2,(X2+X,Y2+Y),15,1
160 NEXT
170 GOTO 60
```

LINEA 10 SELECCIONA LA PANTALLA, TAMAÑO NORMAL, FONDO Y BORDE NEGROS, TEXTO BLANCO.
LINEA 30 LIMPIA LA PANTALLA.

LINEA 40 EL SPRITE 0 ES LA ESTRELLA.
LINEA 50 EL SPRITE 1 ES EL DISEÑO DEL PLANETA.
LINEA 60 BUCLE.
LINEA 70 MOVER LA ESTRELLA CON X DE 1.5 EN 1.5.
LINEA 80 MOVER LA ESTRELLA CON Y DE 1 EN 1.
LINEA 90 X1 = X-COORDENADAS PLANETA 1.
LINEA 100 Y1 = Y-COORDENADAS PLANETA 1.
LINEA 110 X2 = X-COORDENADAS PLANETA 2.
LINEA 120 Y2 = Y-COORDENADAS PLANETA 2.
LINEA 130 MUEVE LA ESTRELLA A UNA NUEVA POSICION.
LINEA 140 MUEVE EL PLANETA 1 A UNA NUEVA POSICION.
LINEA 150 MUEVE EL PLANETA 2 A UNA NUEVA POSICION.
LINEA 160 SIGUIENTE BUCLE.
LINEA 170 COMIENZO DEL BUCLE DE NUEVO.

Choque de *sprites*

El TMS 9918/9929 VDP puede detectar colisiones de *sprites*. Es muy útil para los juegos con choques de *sprites*. Hay dos instrucciones BASIC que permiten la detección de los choques: ON SPRITE GOSUB y SPRITE ON/OFF/STOP.

La instrucción ON SPRITE GOSUB fija la subrutina de interrupción por choque de *sprites*. Indica al ordenador en qué línea comienza la subrutina de tratamiento de dichas interrupciones.

La instrucción SPRITE ON activa la detección de los choques haciendo saltar el programa a la subrutina especificada por la instrucción ON SPRITE GOSUB, cuando choquen dos *sprites*. Has de ejecutar la orden SPRITE ON para poder detectar las colisiones.

Si un punto de dos *sprites* se superpone, se considerará como una colisión. Incluso si los dos puntos superpuestos son transparentes.

Aunque el ordenador puede detectar los choques, no puede decirte qué *sprites* han chocado, ni tampoco dónde se produjo la colisión.

Cuando haya una interrupción, o sea, dos *sprites* colisionan, se ejecuta automáticamente SPRITE STOP. Así se evita que se llame de nuevo a la subrutina de tratamiento, mientras se esté tratando la colisión actual y hubiese otro nuevo choque. Pero recuerda que si hubo otro choque durante la ejecución de la subrutina, saltará inmediatamente a dicha subrutina una vez haya ejecutado por completo la actual, a menos que dentro de la subrutina inhibas la detección de colisiones mediante SPRITE OFF.

Después de abandonar la subrutina de tratamiento el ordenador ejecutará automáticamente SPRITE ON, permitiendo otra interrupción, excepto si en la subrutina incluiste un SPRITE OFF.

La instrucción SPRITE OFF desactiva la detección de colisiones.

Ejemplo:

En este ejemplo verás dos *sprites* cuadrados: uno amarillo y otro blanco, acercándose el uno al otro desde lados opuestos de la pantalla. Cuando choquen,

la instrucción ON SPRITE GOSUB se activará y hará que la ejecución continúe en la subrutina de tratamiento. La instrucción SPRITE OFF, dentro de la rutina de interrupción de *sprites*, evita la detección de más choques. Incluimos esta orden ya que los dos *sprites* continuarán chocando después de la ejecución de la subrutina.

Si no se ejecutase dicha instrucción, el ordenador ejecutaría de forma infinita la subrutina de tratamiento (prueba esta rutina sin SPRITE OFF y observa lo que sucede).

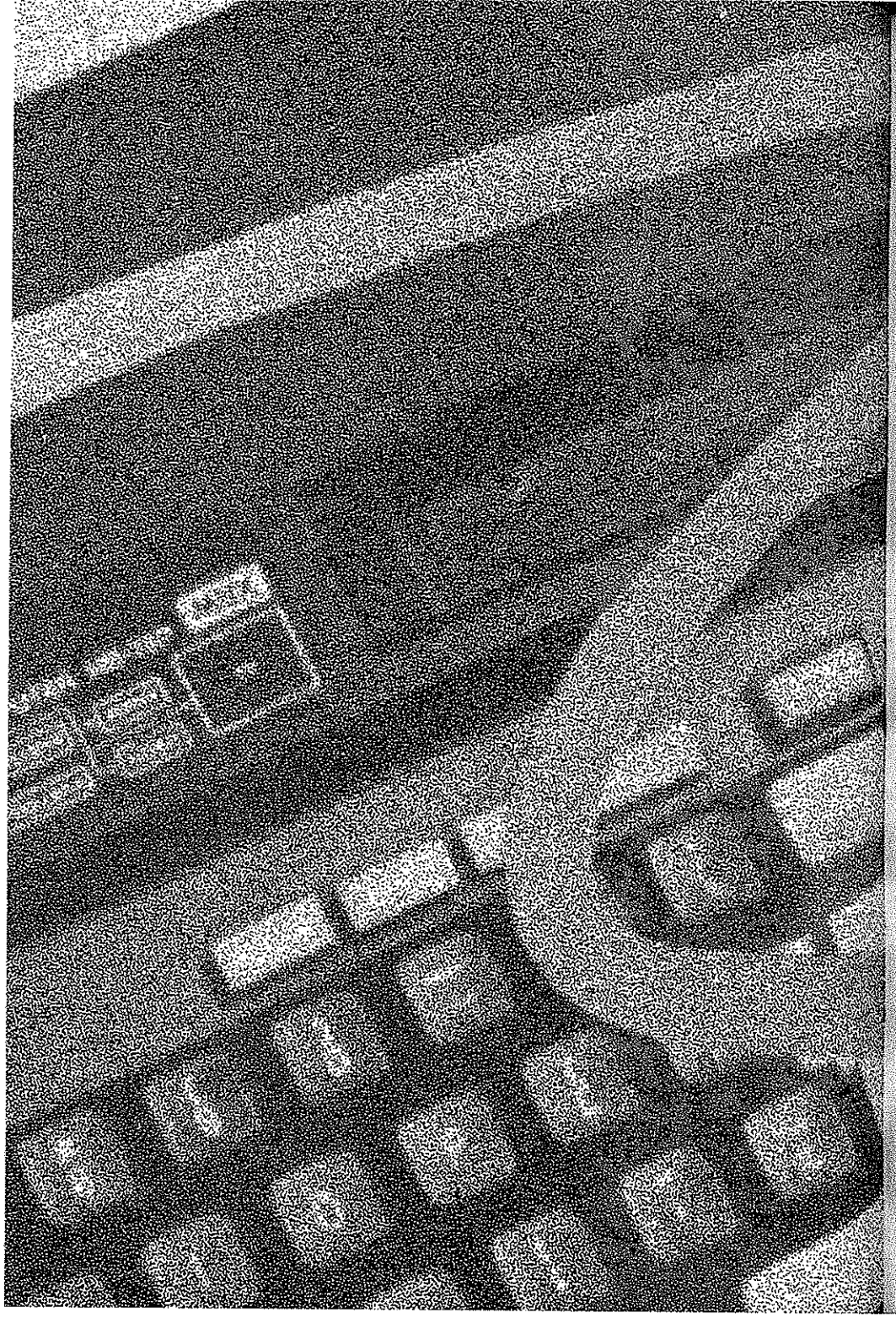
```
10  ON SPRITE GOSUB 110
20  SCREEN 2,0
30  SPRITE$(0)=STRING$(8,CHR$(255))
40  SPRITE$(1)=STRING$(8,CHR$(255))
50  SPRITE ON
60  FOR I=10 TO 240
70  PUT SPRITE 0,(I,100),11,0
80  PUT SPRITE 1,(250-I,100),15,1
90  NEXT I
100 GOTO 50
105 REM Rutina de choque de sprites
110 SPRITE OFF
120 BEEP
130 RETURN
```

LINEA 10 FIJA LA SUBRUTINA EN LA LINEA 110.
LINEA 20 PANTALLA GRAFICA.
LINEA 30 EL SPRITE 0 ES UN CUADRADO.
LINEA 40 EL SPRITE 1 ES UN CUADRADO.
LINEA 50 ACTIVA LA DETECCION DE COLISION DE SPRITES.
LINEA 60 BUCLE.
LINEA 70 SPRITE (AMARILLO) SE MUEVE DESDE LA IZQUIERDA.
LINEA 80 SPRITE (BLANCO) SE MUEVE DESDE LA DERECHA.
LINEA 90 SIGUIENTE BUCLE.
LINEA 100 COMIENZA EL NUEVO BUCLE.
LINEA 110 INHIBE LA DETECCION (CON UNA SOLA VEZ BASTA).
LINEA 120 PITIDO DE AVISO.
LINEA 130 VUELVE AL BUCLE.

Notas sobre colisión de *sprites*

Cada vez que se ejecute una instrucción, el ordenador analiza si se ha producido una colisión de *sprites*. No se puede decir que se busque un choque de *sprites* en un determinado instante.

La función VDP te puede dar el estado del choque de *sprites* del registro 8 del procesador de video. (Véase la parte correspondiente al VDP.)



44

Gráficos avanzados (V)

Acceso al procesador de video

Función VDP

El VDP TMS 9929 A tiene ocho registros de sólo escritura, y un registro de sólo lectura. Puedes acceder a estos registros desde el BASIC mediante la función VDP.

A los registros de sólo escritura del VDP(0) al VDP(7) únicamente les puedes asignar valores; por ejemplo, VDP(1)=2. No hay modo de leer estos registros directamente desde el VDP.

Sin embargo, el ordenador guarda una copia de estos registros en el área de trabajo del sistema, en las posiciones RG0SAV a RG7SAV. (Véase la parte de la RAM del sistema, tema de las BIOS.)

VDP(8) es el registro de sólo lectura. Para leer el contenido de registro escribe:

```
PRINT VDP(8)
```

Recuerda que si asignas un número erróneo a estos registros la pantalla se podría bloquear. Si te sucede esto debes pulsar RESET o apagar el ordenador para inicializar el VDP. Antes de utilizar la función VDP asegúrate de lo que haces.

Registros del VDP

Registros de sólo escritura

Registro 0 ... VDP(0)

El registro 0 tiene dos bits de control, BIT 1(M3) y BIT 0(EV). Del BIT 2 al BIT 7 deben ser 0.

BIT 0 EV: VDP externo. Posibilita la entrada de datos desde otro VDP externo. EV = 0 en casi todos los MSX.
0 = inactivo
1 = activo

BIT 1 M3: Bit de modo 3. El M3 se utiliza para seleccionar el modo de pantalla conjuntamente con el registro 1 (véase registro 1).

RO

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	M3	EV=0

Registro 1 ... VDP(1)

El registro 1 tiene 7 bits de control del VDP. El BIT 2 está reservado, y debe estar siempre a 0.

BIT 0 MAG: Ampliación de *sprites*.
0 = Sin ampliar resolución de 1 × 1 puntos.
1 = Ampliados, resolución de 2 × 2 puntos.

BIT 1 SIZE: Selector del tamaño *sprites*
0 = *Sprites* de 8 × 8 puntos.
1 = *Sprites* de 16 × 16 puntos.

BIT 2 Reservado = 0

BIT 3 M2: Bit de modo 2. El M2 se emplea para seleccionar el modo de pantalla (véase bit 4).

BIT 4 M1: Bit de modo 1. El M1 se utiliza para seleccionar el modo de pantalla. La combinación de M1, M2 y M3 proporciona dicho modo.

M1	M2	M3	
0	0	0	Modo 1 de texto
0	0	1	Modo 2 de alta resolución gráfica
0	1	0	Modo 3 de gráficos multicolor.
0	1	1	Modo 0 de texto

BIT 5 IE: Activa las interrupciones.
0 = Desactiva las interrupciones del VDP.
1 = Activa las interrupciones del VDP.

BIT 6 BL: Activa/desactiva la pantalla. Te permite desconectar la pantalla.

0 = Desactiva la pantalla poniéndola del color del borde.
1 = Activa la pantalla.

BIT 7 VRAM: Selecciona el tipo de RAM de video a utilizar (VRAM = 1 es lo normal en el MSX).
0 = 4K RAM
1 = 16K RAM

R1

B7	B6	B5	B4	B3	B2	B1	B0
VRAM	BL	IE	M1	M2	0	SIZE	MAG

Registro 2 ... VDP(2)

El registro 2 define la dirección base de la tabla de nombres. Su rango se encuentra entre 0 y 15. Este registro tiene los 4 bits más significativos de los 14 de la dirección de la tabla de nombres. Por tanto, debes dividir por &H400 la dirección.

R2

B7	B6	B5	B4	B3-	B2	B1	B0
0	0	0	0	DIRECCION DE LA TABLA DE NOMBRES			

Nota: R2*&H400 = DIRECCION DE LA TABLA DE NOMBRES

Registro 3 ... VDP(3)

El registro 3 define la dirección base de la tabla de color. Su rango se encuentra entre 0 y 255. Este registro representa a los 8 bits más significativos de los 14 de la dirección de la tabla de color. Por tanto, deberás escribir la dirección dividida por &H40.

R3

B7	B6	B5	B4	B3	B2	B1	B0
DIRECCION DE LA TABLA DE COLOR							

Nota: R3*&H40 = DIRECCION DE LA TABLA DE COLOR

Registro 4 ... VDP(4)

El registro 4 define la dirección base de la tabla generadora de modelos, texto o multicolor, dependiendo del modo de pantalla.

Su rango se encuentra entre 0 y 7. Este registro representa los 3 bits más significativos de los 14 de la dirección de la tabla generadora de modelos/texto/multicolor. Por tanto, has de escribir la dirección dividida por &H800.

R4

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	DIRECCION TABLA D./T./M.		

Nota: R4*&H800 = DIRECCIONES DE LA TABLA GENERADORA DE MODELOS/TEXTO/MULTICOLOR

Registro 5 ... VDP(5)

El registro 5 define la dirección base de la tabla de atributos de *sprites*. Su rango se encuentra entre 0 y 127. Este registro representa los 7 bits más significativos de los 14 de la dirección de la tabla de atributos de *sprites*. Por tanto, debes escribir la dirección dividida por &H80.

R5

B7	B6	B5	B4	B3	B2	B1	B0
0	DIRECCION TABLA DE ATRIBUTOS DE *SPRITES*						

Nota: R5*&H80 = DIRECCION DE LA TABLA DE ATRIBUTOS DE *SPRITES*.

Registro 6 ... VDP(6)

El registro 6 define la dirección base de la tabla generadora de modelos de *sprite*. Su rango se encuentra entre 0 y 127. Este registro representa los 3 bits más significativos de los 14 de la dirección de la tabla generadora de diseños de *sprite*. Por tanto, debes escribir la dirección dividida por &H800.

R6

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	DIRECCION DE LA TABLA GEN. MO. SP.		

Nota: R6*&H800 = DIRECCION DE LA TABLA GENERADORA DE DISEÑOS DE *SPRITE*

Registro 7 ... VDP(7)

El registro 7 está dividido en dos partes. Los 4 bits más significativos contienen el color de la tinta en modo texto, mientras que los 4 menos significativos tienen el color del fondo en los modos texto y gráfico.

El rango para el color del texto está entre 0 y 15, al igual que para el color de fondo.

R7 = <color del texto>*&H10 + <color del fondo>

R7

B7	B6	B5	B4	B3	B2	B1	B0
COLOR DE LA TINTA				COLOR DEL FONDO			

Registro de sólo lectura

El registro de estado ... VDP(8)

Este registro tiene el *flag* pendiente de interrupción, el *flag* de colisión de *sprites* y el *flag* del quinto *sprite*.

***Flag* de interrupción F: BIT 7**

El *flag* de interrupción se activa (1) cuando ha terminado el análisis de pantalla. Se pone a 0 después de leer el registro de estado, o cuando se inicializa el VDP desde fuera.

***Flag* de colisión de *sprites* C: BIT 5**

El VDP analiza si dos o más *sprites* se superponen en un punto. Este análisis se hace cada 1/50 segundos.

1 = colisión de *sprites*
0 = no hay superposición.

***Flag* del quinto *sprite* 5S: BIT 6**

El 5S se activa cuando hay más de cuatro *sprites* en una línea horizontal (líneas 0 a 192). El VDP no puede tener más de cuatro *sprites* en una sola línea. Si hay más de cuatro *sprites* no se visualizarán a partir del quinto.

Número del quinto *sprite*

Si hay más de cuatro *sprites* en una línea horizontal y 5S se activa, entonces el número de *sprite* que ha roto la regla se guarda en los 5 bits menos significativos de este registro.

R2

B7	B6	B5	B4	B3	B2	B1	B0
F	5S	C	NUMERO DEL QUINTO *SPRITE*				

45

Gráficos avanzados (VI)

La RAM de video

El MSX posee una RAM de video de 16K separada de la memoria principal. El procesador de video la trata y refresca; es independiente del procesador central Z-80 (la RAM dinámica utilizada por el MSX ha de ser refrescada para no perder los datos almacenados).

Esto tiene las siguientes ventajas:

1. El procesador Z-80 no necesita refrescar a la RAM de pantalla, por lo que ahorra tiempo.
2. La pantalla y los gráficos no utilizan la memoria principal utilizada por el Z-80, por lo que el procesador central tiene más memoria.

La RAM de video está dividida en secciones; cada una desempeña una única función.

Es posible acceder a la RAM de video desde el BASIC mediante las siguientes instrucciones y funciones:

BASE	Proporciona la dirección base de varias tablas de la RAM de video.
VPEEK	Lee el contenido de una posición de la RAM de video.
VPOKE	Escribe en una posición de la RAM de video.

La función BASE da la dirección base actual de todas las tablas de pantalla de la RAM de video (para más detalles, véase BASE).

Nombre de tabla y dirección en hex.	Modo de texto 0	Modo de texto 1	Modo gráfico 2	Modo gráfico 3
Tabla de modelos	&H0000	&H1800	&H1800	&H0800
Tabla de color		&H2000	&H2000	
Tabla generadora de modelos	&H0800	&H0000	&H0000	&H0000
Tabla de atributos de *sprites*		&H1B00	&H1B00	&H1B00
Tabla generadora de *sprites*		&H3800	&H3800	&H3800

46

Efectos de sonido con el PSG

El generador de sonido del MSX, AY-3-8910, lleva en el mercado bastante tiempo. Fabricado por General Instruments a finales de los 70, es todavía uno de los *chips* de sonido más populares. Se utiliza mucho en los sistemas de ordenadores de 8 bits y en las máquinas de juegos. La razón de su éxito es su flexibilidad y facilidad para generar tres voces, con sonido periódico, y un canal de sonido puro, simultáneamente. También porque tiene un microprocesador que no interrumpe al procesador central mientras ejecuta la música. Aunque actualmente está algo desfasado, puedes sacarle un gran partido.

Esta parte trata sobre el funcionamiento del generador de sonido AY-3-8910 y de la construcción SOUND.

Como ejemplo, prueba el siguiente programa que genera un fantástico ruido de reactor

```
10 SOUND 0,87
20 SOUND 7,62
30 SOUND 8,16
40 SOUND 11,179
50 SOUND 12,45
60 SOUND 13,14
```

Formas de onda generadas por el PSG

Básicamente el PSG sólo puede generar dos tipos de formas de onda: una onda periódica cuadrada y un sonido uniforme (llano).

Ejemplo de onda periódica:

```
10 SOUND 7,62
20 SOUND 8,15
30 SOUND 1,1
40 SOUND 0,28
```

Ejemplo de sonido uniforme:

```
10 SOUND 7,55
20 SOUND 8,15
```

Estas ondas se pueden combinar y modificar para formar ruidos complejos, intentando sonidos similares a los reales. Para ello debes controlar la frecuencia, el tono y la forma de onda de cada canal de sonido mediante la instrucción SOUND, con la que se accede al PSG.

Registros de sonido PSG

El AY-3-8910 PSG tiene catorce registros en los que puedes escribir utilizando la instrucción SOUND. La sentencia SOUND posee la siguiente sintaxis:

SOUND <número de registro>,<valor asignado>

Asignando valores adecuados a estos registros puedes seleccionar los canales, ondas de sonido, frecuencia de tono, sonido y volumen.

Hay catorce registros, con las siguientes funciones:

REGISTRO	DESCRIPCION
0	Ajuste fino de frecuencia del canal A.
1	Ajuste básico de frecuencia del canal A.
2	Ajuste fino de frecuencia del canal B.
3	Ajuste básico de frecuencia del canal B.
4	Ajuste fino de frecuencia del canal C.
5	Ajuste básico de frecuencia del canal C.
6	Frecuencia del generador de ruido.
7	Mezclador de los canales A, B y C.
8	Control del volumen en el canal A.
9	Control del volumen en el canal B.
10	Control del volumen en el canal C.
11	Período de envolvente (bajo).
12	Período de envolvente (alto).
13	Modelo de onda.

Elección del sonido de cada canal: registro 7, el mezclador

Este es el registro que activa o desactiva el sonido de los tres canales. Puede seleccionar la fuente de la onda de sonido desde el generador de sonido, o desde el generador de tono individual de cada canal.

Los bits 5, 4 y 3 del registro 7 activan el generador de sonido de los canales C, B y A, respectivamente. Con 1 está desconectado y con 0 está conectado. Los bits 2, 1 y 0 del registro 7 ponen en funcionamiento el generador de tono de los canales C, B y A, respectivamente. Por ejemplo, si quieres que el canal C emita un sonido, el bit 2 debe estar a 0.

BIT	B7	B6	B5	B4	B3	B2	B1	B0
	/	/	RUIDO			TONO		
	/	/	C	B	A	C	B	A
	0	0	1	1	1	0	1	1
			OFF	OFF	OFF	ON	OFF	OFF

El formato de la sentencia SOUND, según se encuentra reflejado arriba, sería:

SOUND 7,&00111011; o en decimal: SOUND 7,59

El tono del canal A depende de los registros 0 y 1; lo veremos más adelante.

Si deseas que el canal B emita ruido blanco, y C y A emitan según el formato anterior, tendrás que cambiar el registro 7 de la siguiente manera:

BIT	B7	B6	B5	B4	B3	B2	B1	B0
	/	/	RUIDO			TONO		
	/	/	C	B	A	C	B	A
	0	0	1	0	1	0	1	1
			OFF	ON	OFF	ON	OFF	OFF

El formato de la sentencia SOUND, según se encuentra reflejado arriba, sería:

SOUND 7,&B00101011; o en decimal: SOUND 7,43

Puedes hacer que un canal emita un ruido uniforme y un tono al mismo tiempo activando ambos generadores. Si deseas que todos los canales emitan tono y ruido, obtendrás:

BIT	B7	B6	B5	B4	B3	B2	B1	B0
	/	/	RUIDO			TONO		
	/	/	C	B	A	C	B	A
	0	0	0	0	0	0	0	0
			ON	ON	ON	ON	ON	ON

El formato de la sentencia SOUND, según se encuentra reflejado arriba, sería:

SOUND 7,&B00000000; o en decimal: SOUND 7,0

Nota: Los bits B7 y B6 no se utilizan, y por eso se encuentran a 0.

Generador de tono: registros 0, 1, 2, 3, 4 y 5

Para fijar la frecuencia del generador de tono de cada canal escribe en los registros 0, 1, 2, 3, 4 y 5. Cada canal tiene dos registros que fijan la frecuencia de tono a generar.

Registro 0 y 1 para el canal A.
Registro 2 y 3 para el canal B.
Registro 4 y 5 para el canal C.

REGISTRO	RANGO	DESCRIPCION
0	0-255	Ajuste fino de la frecuencia del canal A.
1	0-15	Ajuste básico de la frecuencia del canal A.
2	0-255	Ajuste fino de la frecuencia del canal B.
3	0-15	Ajuste básico de la frecuencia del canal B.
4	0-255	Ajuste fino de la frecuencia del canal C.
5	0-15	Ajuste básico de la frecuencia del canal C.
6	0-31	Frecuencia del generador de sonido.

La frecuencia (f) se calcula como:

f = (registro 0) + (registro 1) × 256

Esta fórmula no proporciona la frecuencia en Hz, pero cuanto mayor sea la frecuencia menor será el tono.

La frecuencia más alta se genera como SOUND 0,0: SOUND 1,0, que proporciona una f = 0; y la frecuencia más baja se dará cuando SOUND 0, 255: SOUND 1,15.

Para obtener la frecuencia generada en Hz utiliza la siguiente fórmula:

$$f = \frac{178900}{(16 \times Hz)}$$

178900 es la frecuencia del reloj del PSG (que será distinta en el modelo británico del MSX).

Si quieres que la frecuencia que genere el PSG sea exacta debes utilizar el cálculo anterior. Pero no lo hagas con una calculadora; utiliza en su lugar el ordenador.

```
10 INPUT "FRECUENCIA EN HERTZIOS";HZ
20 F=INT(1789800#/(16*HZ))
30 PRINT "TONO FINO,REGISTROS 0,2 o 4:";F MOD 256
40 PRINT "TONO,REGISTROS 1,3, o 5:";INT(F/256)
```

Este programa te hará los cálculos.

Ejemplo:
Genera sonido de 2000 Hz por el canal A.

f = 1789800/(16*2000) = 55.9

```
10 SOUND 0,255
20 SOUND 1,0
30 SOUND 7,62     ;MEZCLADOR
40 SOUND 8,10     ;VOLUMEN
```

Frecuencia del generador de sonido: registro 6

Para cambiar la frecuencia del generador de sonido uniforme escribe en el registro 6 del PSG.

Su rango se encuentra comprendido entre 0 y 31; cuanto más bajo sea el valor, más alta será la frecuencia.

Ejemplo:

```
10 SOUND 7,55              ;CONECTA EL CANAL A
20 SOUND 8,15              ;VOLUMEN DEL CANAL 1 AL 15
30 FOR F=0 TO 31
40 SOUND 6,F
50 FOR I=1 TO 200: NEXT    ;RETARDO
60 NEXT F
```

Oirás un sonido uniforme con la frecuencia disminuyendo.

Volumen: registros 8, 9, 10

El volumen del sonido emitido por los canales A, B y C se controla con los registros 8, 9 y 10, respectivamente.

sin sonido máximo volumen
0 <----------> 15
REGISTRO 8 = CANAL A
REGISTRO 9 = CANAL B
REGISTRO 10 = CANAL C

Cuando a estos registros se les asigne el valor 16, se activa la onda; o sea, el sonido varía su volumen según la forma de la onda y la frecuencia. La onda se define mediante los registros 11, 12 y 13.

Recuerda que el volumen del altavoz del televisor depende de su control de volumen. Si pones el volumen al mínimo no oirás nada. El volumen de sonido del MSX también depende del amplificador externo.

Las ondas: registros 11, 12 y 13

Cuando no se emplean ondas el volumen del sonido permanece constante en el nivel fijado por los registros 8, 9 y 10. Esto es bastante aburrido; todo lo que oirás será un ruido de frecuencia constante. Sin embargo, el AY-3-8910 PSG te ofrece una amplia gama de ondas para cambiar periódicamente el volumen, de acuerdo con la forma de la onda elegida.

Cuando pones los registros de control del volumen 8, 9 y 10 a 16, activas las ondas. Al activar las ondas el volumen del sonido cambia según los valores de los registros 11, 12 y 13.

		SIN ONDA	CON ONDA
CANAL A	REGISTRO 8	0-15	16
CANAL B	REGISTRO 9	0-15	16
CANAL C	REGISTRO 10	0-15	16

La forma de la onda se selecciona poniendo en el registro 13 un número comprendido entre 0 y 15. El PSG tiene duplicadas algunas formas de onda, por lo que sólo tienes ocho formas de onda diferentes.

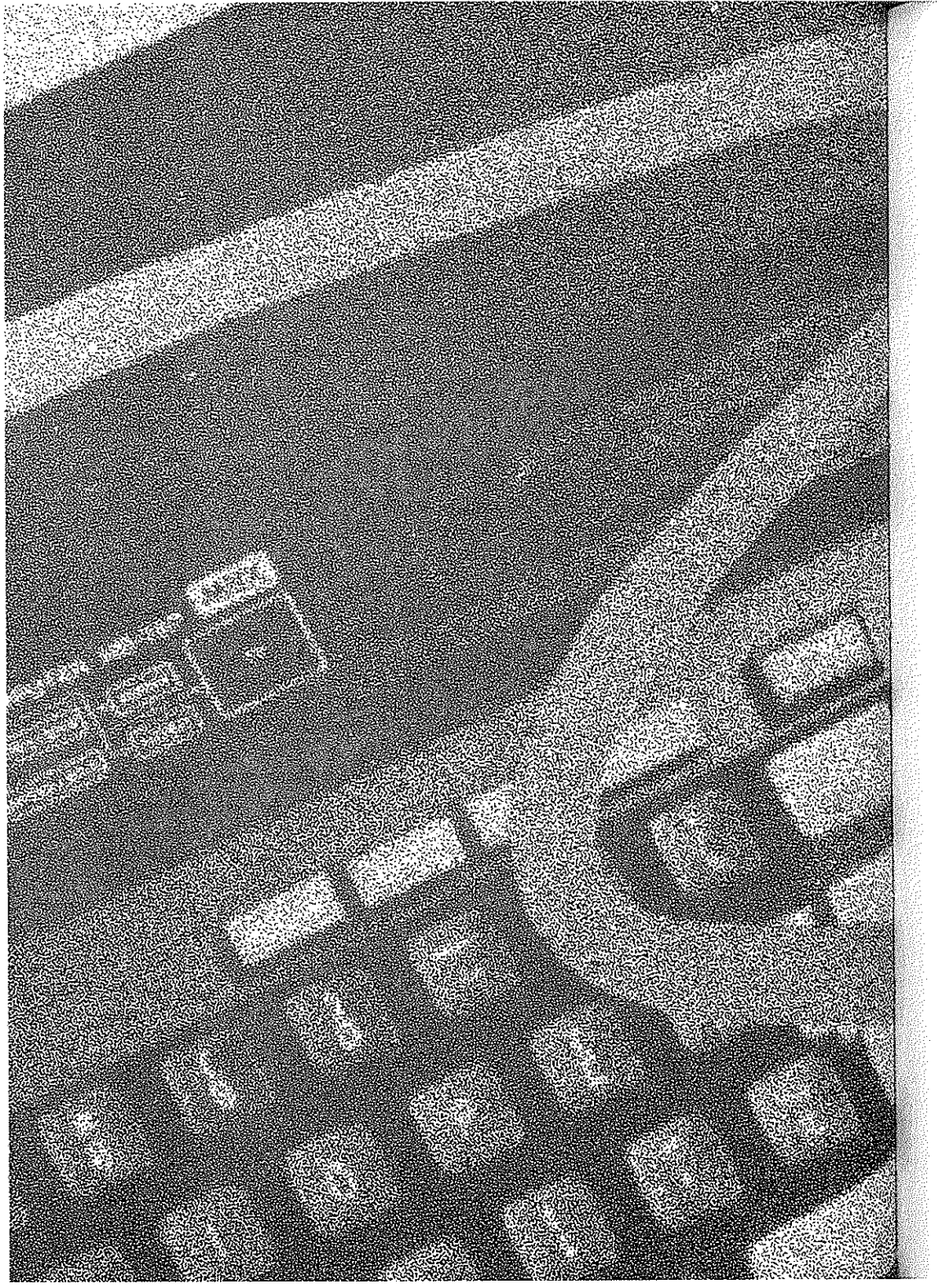
La frecuencia de cambio del volumen, o sea, el período de la onda, lo has de fijar en los registros 11 y 12; en el 12 ajustas el período y en el 11 proporcionas un ajuste más fino.

	RANGO
REGISTRO 11	0-255
REGISTRO 12	0-255

Período = (registro 12) + 256*(registro 11)

Sólo puedes tener una onda en cada instante, y los tres canales han de usar la misma forma de onda.

(Consultar SOUND en la *Guía de referencia del BASIC* para ver las formas de las ondas.)



47

Manejo de ficheros

En el BASIC del MSX es posible grabar los datos, como el contenido de matrices de tipo cadena de caracteres, en cassette, y recíprocamente para leer de él. A este tipo de estructura se le denomina "fichero". Hay instrucciones y funciones BASIC que te permiten escribir y leer desde el periférico indicado.

MAXFILES

Antes de emplear un fichero es una buena idea aumentar el valor de MAXFILES, función utilizada para fijar el número máximo de ficheros permitidos a la vez. El valor por defecto de MAXFILES es 1, pero normalmente no es suficiente, ya que sólo te permite trabajar con el cassette. El número máximo de MAXFILES es 15.

Técnicamente, MAXFILES aumenta el tamaño de bloque de control de ficheros en memoria. El bloque de control de ficheros se emplea como área de trabajo por la ROM del MSX cuando se trabaja con ficheros. (Véase el mapa de memoria. Tema 48.)

OPEN

Para indicar al ordenador que se va a leer o escribir de ficheros debes emplear la orden OPEN, con la que se abre un determinado periférico. Prepara un *buffer* en el bloque de control de ficheros y fija el modo de E/S del *buffer*.

Antes de ejecutar cualquier instrucción o función de esta lista has de ejecutar la orden OPEN:

PRINT#	PRINT# USING
INPUT#	LINE INPUT#
INPUT$(#)	EOF

OPEN "<nombre del periférico>[<nombre del fichero>]" [FOR <modo>] AS [#] <número de fichero>

Hay cuatro dispositivos básicos en la presente versión del MSX; son:

CAS:	cassette.
CRT:	pantalla de texto.
GRP:	pantalla gráfica.
LPT:	impresora.

De ellos sólo CAS: y GRP: tienen verdadera utilidad. CAS: se especifica cuando se utiliza el cassette para almacenar o recuperar datos, y GRP: se especifica cuando escribes en las pantallas gráficas. No es posible ejecutar PRINT en las pantallas de MODO 2 y 3; por tanto, para escribir en ellas has de hacerlo a través de un fichero. En el tema 42 tienes una detallada descripción de cómo emplear los ficheros para escribir en las pantallas gráficas.

Hay tres modos de E/S:

OUTPUT	Especifica la forma de salida secuencial, o sea, escribiendo en un periférico mediante PRINT# y PRINT# USING.
INPUT	Especifica una forma de entrada secuencial; por ejemplo, leer de una unidad utilizando INPUT#, etc.
APPEND	Especifica la forma de añadir secuencialmente.

<número de fichero> es un entero cuyo valor está comprendido entre uno y MAXFILES, el número máximo de ficheros. Los números de ficheros se dan en sentencias PRINT#; por ejemplo, el ordenador sabe de qué unidad debe leer o escribir. El <número de fichero> permanece asociado al fichero mientras está abierto. No puede ser uno que ya esté usado en el programa.

Ejemplo:

Para abrir un fichero de cassette en el que queremos escribir información:

```
OPEN "CAS: INFO" FOR OUTPUT AS#1
```

CLOSE

Es la instrucción opuesta a OPEN. Una buena táctica es la de cerrar los ficheros tan pronto como hayas acabado de trabajar con ellos

PRINT#

Para escribir en varias unidades sin cambiar el formato de la instrucción se ha creado la instrucción PRINT#. Al signo "#" le sigue el número de fichero especificado en la orden OPEN.

Cuando se utiliza como periférico el cassette, PRINT# tiene el efecto de registrar la información dada.

PRINT# USING

Esta es una versión de PRINT USING para ficheros y tiene el mismo efecto; se emplea principalmente para escribir en las pantallas gráficas con un formato específico.

INPUT#

Para leer datos de otras unidades que no sea el teclado, como el cassette, has de emplear la instrucción INPUT#. Tiene el mismo efecto que la sentencia INPUT, pero en vez de tomar la información del teclado la lee del periférico especificado en la orden OPEN; en la mayoría de los casos la unidad será el cassette.

En las lecturas de datos numéricos se ignoran todos los espacios en blanco, marcas de fin de línea y saltos de línea. Análogamente sucede con las cadenas de caracteres, en las que también se ignoran las comillas.

INPUT$(#)
LINE INPUT#

Son las versiones de INPUT$ y LINE INPUT para los ficheros. (Véase INPUT#.)

EOF

EOF analiza si se ha llegado al final del fichero. Si así fuese EOF devuelve el valor − 1; en caso contrario devolverá el valor 0.

Programa ejemplo

En este programa de demostración te damos una idea de cómo tratar los ficheros.

El programa te pide cinco nombres y los almacena en un fichero de un cassette. Entonces te pregunta si quieres cargarlos otra vez.

```
10  DIM N$(5)
20  MAXFILES = 4
30  FOR I=1 TO 5
40  INPUT "DAME UN NOMBRE";N$
50  NEXT I
60  PRINT "ESTA ENCENDIDO EL CASETE? (s=SI)"
```

```
70  IF INKEY$<>"s" THEN 70
80  OPEN "CAS:" FOR OUTPUT AS #2
90  FOR I=1 TO 5
100 PRINT#2,N$(I)
110 NEXT I
120 CLOSE#2
130 PRINT "ESTAS PREPARADO PARA CARGAR?(s=SI)"
140 IF INKEY$<>"s" THEN 140
150 OPEN "CAS:" FOR INPUT AS #3
160 FOR I=1 TO 5
170 INPUT#3,N$(I)
180 NEXT I
190 CLOSE#3
200 FOR I=1 TO 5
210 PRINT N$(I)
220 NEXT I
```

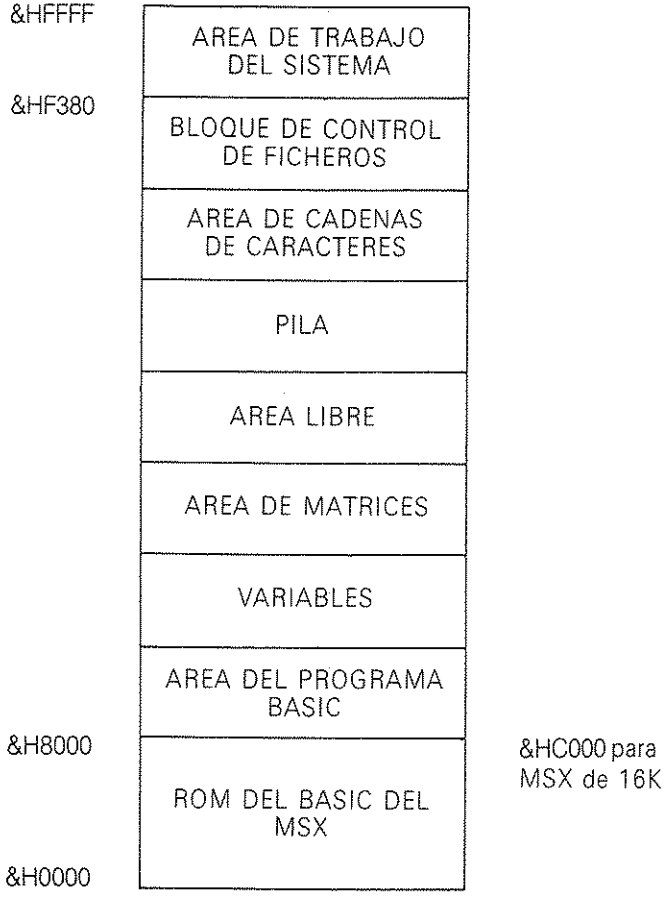
LINEA 10 MATRIZ PARA NOMBRES.
LINEA 20 AUMENTA EL NUMERO MAXIMO DE FICHEROS.
LINEA 30 ENTRA CINCO NOMBRES PARA PODER HACER ALGO CON ELLOS.
LINEA 60 CASSETTE CONECTADO.
LINEA 70 ESPERA A QUE CONECTES EL CASSETTE.
LINEA 80 ABRE UN FICHERO EN EL CASSETTE.
LINEA 90 SALIDA PARA EL CASSETTE MEDIANTE PRINT#.
LINEA 120 CIERRA EL FICHERO CUANDO ACABA.
LINEA 130 REBOBINA EL CASSETTE PARA QUE PODAMOS CARGAR LOS DATOS GRABADOS (PLAY).
LINEA 150 ABRE UN FICHERO PARA ENTRADA
LINEA 160 LEE DEL FICHERO DE CASSETTE UTILIZANDO INPUT#.
LINEA 190 CUANDO SE ACABE CIERRA EL FICHERO.
LINEA 200 ESCRIBE LOS RESULTADOS.


```
RUN
DAME UN NOMBRE? CELSO
DAME UN NOMBRE? TOMAS
DAME UN NOMBRE? PEPITA
DAME UN NOMBRE? JUANITO
DAME UN NOMBRE? ANTONIO
ESTA ENCENDIDO EL CASSETTE? (s=SI)        PULSA RECORD
ESTAS PREPARADO PARA CARGAR?(s=SI REBOBINA Y PULSA PLAY
CELSO
TOMAS
PEPITA
JUANITO'
ANTONIO

OK
```

48

Mapa de memoria



Area de trabajo

El AREA DE TRABAJO está entre las posiciones de memoria &HFFFF y la &HF380; la RAM emplea esta área para sus operaciones internas. Esta área soporta las variables del sistema (en la sección de referencia BIOS te damos una lista con todas ellas).

Bloque de control de ficheros

Esta área se reserva para las operaciones de entrada y salida con ficheros. Sentencias como PRINT# e INPUT# trabajan con dicho bloque.

Puedes variar el tamaño del bloque de control de ficheros mediante la función MAXFILES.

El límite superior de este bloque está en la dirección &HF380. Puedes reducir dicho límite para crear un área que almacene programas en código máquina y datos. La orden CLEAR reduce este límite si se pone un número menor.

Ejemplo: Crear un área de código máquina del usuario entre las direcciones &HF380 y &HF000:

```
CLEAR, &HF000
```

Area de cadenas de caracteres

En esta área se guardan los contenidos de las variables de tipo cadena de caracteres. El tamaño por defecto del área de las cadenas es de 200 bytes, fijando así un límite de 200 caracteres de longitud para las variables de este tipo.

Sin embargo, puedes aumentar dicha longitud mediante la instrucción CLEAR.

Ejemplo: Incrementar el área de cadenas de caracteres a 255 bytes:

```
CLEAR 255
```

Pila

Aquí se almacenan las pilas de los bucles FOR/NEXT y las llamadas GOSUB. Guarda las direcciones de retorno a las que debe volver el BASIC al encontrarse una sentencia NEXT o RETURN.

Area Libre

Es el área que no se utiliza. Puedes conocer su tamaño mediante la función FRE, del siguiente modo:

```
PRINT FRE(0)
28815          en sistemas MSX de 32K y 64K recién encendidos.
12431          en sistemas MSX de 16K recién encendidos.
```

Observa que esta área disminuye al aumentar tu programa BASIC, así como con el número de variables.

Area de matrices

Esta área almacena las matrices declaradas mediante DIM, incrementándose cada vez que se ejecute una instrucción DIM. Si la matriz es de tipo cadena de caracteres, se guardarán los punteros a los elementos almacenados en el área de cadenas de caracteres.

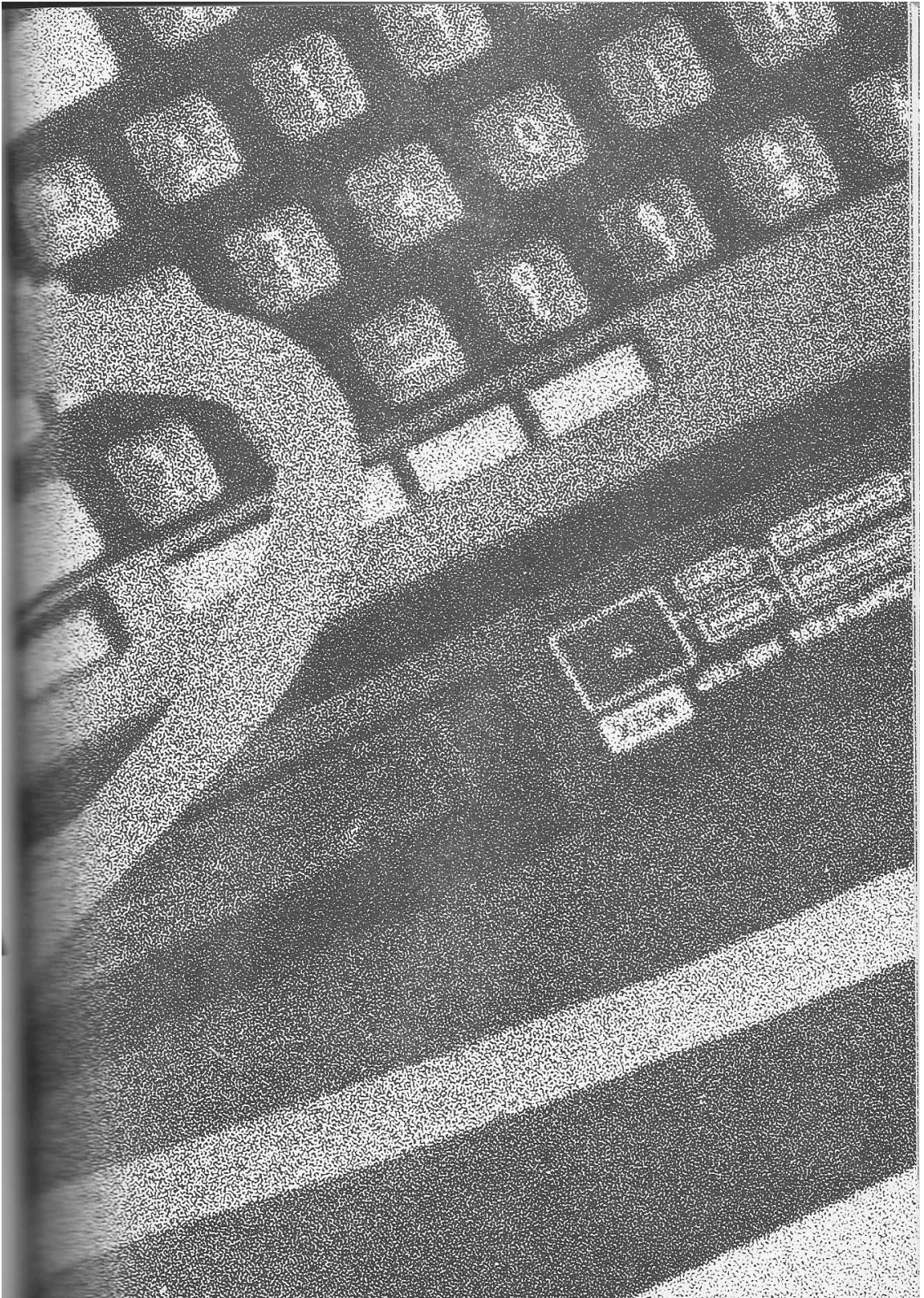
Area de variables

En esta área se almacenan las variables numéricas y los punteros al área de cadenas de caracteres.

La dirección en donde se almacena el dato de una variable en concreto se puede obtener mediante la función VARPTR. (Véase VARPTR en la *Guía de referencia del BASIC.)*

Area del programa BASIC

Aquí se guarda tu programa BASIC.



49

Función USR y código máquina

Has de emplear la función USR para poder ejecutar subrutinas en código máquina desde el BASIC.

La función USR llama al programa en código máquina que comienza en la posición especificada por la instrucción DEF USR. Puedes tener hasta diez rutinas en código máquina simultáneamente. Pero, redefiniendo la función USR, podrás tener más de diez.

Definición de funciones USR

Para llamar a una rutina en código máquina, el ordenador debe saber la dirección de comienzo de la subrutina. Emplea la instrucción DEF USR para indicar la dirección de comienzo de las funciones USR.

Ejemplo: Indicar una subrutina en código máquina que comience en &HF000

```
DEF USR 0 = &HF000        (puedes quitar el 0, quedando:
                           DEF USR = &HF000)
```

Cómo ejecutar una rutina en código máquina

La función USR se emplea del mismo modo que cualquier otra función. Con lo que puede ir dentro de una expresión, como las siguientes:

```
X = USR (99)
Y$ = "M/C" + USR 9("abx")
PRINT USR 7(0)
```

Lo anterior ejecuta la subrutina y devuelve un valor según la hayas codificado. Los números o cadenas de caracteres entre paréntesis son los parámetros a pasar al código máquina. Los resultados de la función USR dependen exclusivamente de la rutina que hayas codificado. Has de pasarle un parámetro, aunque la rutina no lo emplee para nada; ésta puede devolver un único valor o nada, según te sea necesario.

Si sólo quieres ejecutar el código máquina sin parámetros de entrada ni de salida, entonces emplea parámetros y variables sin significado; por ejemplo:

```
NADA = USR 0(0)
```

donde NADA y (0) son una variable y un parámetro mudos.

Cómo pasar parámetros desde el BASIC

Con USR <número> (<parámetro>) puedes pasar cualquier tipo de parámetro al código máquina desde el BASIC. El parámetro ha de ir encerrado entre paréntesis. Cuando el MSX ejecute la función analiza el tipo del argumento, comprueba si el parámetro es un número en precisión simple, o número en doble precisión o una cadena de caracteres.

En caso de que haya un parámetro, el código máquina debe saber de qué tipo es y la posición de memoria en donde se encuentra.

Para saber el tipo de parámetro que se ha pasado toma el contenido de la posición &HF663:

```
&HF663 = 2 = 00000010  parámetro entero.
&HF663 = 4 = 00000100  parámetro de precisión simple.
&HF663 = 8 = 00001000  parámetro de doble precisión.
&HF663 = 3 = 00000011  cadena de caracteres.
```

Localización del parámetro pasado.

Entero (&HF663 = 2).

&HF7F8 = <byte menos significativo>.
&HF7F9 = <byte más significativo>.

<parámetro entero> = <byte menos significativo> + 256*<byte más significativo>.

Número de precisión simple (&HF663 = 4):

```
&HF7F6 = ]
&HF7F7 = ]      valor del parámetro almacenado en
&HF7F8 = ]      decimal codificado en binario
&HF7F9 = ]      desde &HF7F6.
```


Número en doble precisión (&HF663 = 8):

```
&HF7F6 = ]
&HF7F7 = ]
&HF7F8 = ]      valor del parámetro
&h F7F9 = ]     almacenado en
&HF7FA = ]      decimal codificado en binario desde
&HF7FB = ]      la posición &HF7F6.
&HF7FC = ]
&HF7FD = ]
```

Cadena de caracteres (&HF663 = 3)

```
&HF7F8 =        ]menor]
&HF7F9 =        ]mayor] ... <dirección 1> dirección del bloque descriptor de la cadena de caracteres.
```

<dirección 1> = longitud de la cadena de caracteres.
<dirección 1> + 1 = menor].
<dirección 1> + 2 = mayor] ... <dirección 2>, localización de la cadena de caracteres.

Parámetros de salida del código máquina

Para devolver un parámetro desde el código máquina al BASIC mete en la posición &HF663 el valor apropiado según el tipo de parámetro y almacénalo en la dirección pertinente antes de salir de la rutina. El BASIC toma el resultado de las posiciones correspondientes y se lo asigna a la variable indicada en tu programa.

50
Memoria y cartucho del MSX

Introducción

La CPU del MSX, el microprocesador Z80, tiene un *bus* de direcciones de 16 bits, pudiendo direccionar hasta 64K bytes de memoria. La memoria de 64K está dividida en cuatro páginas de 16K. Puedes ampliar esta capacidad de memoria añadiendo un segmento a cada una de las páginas. Un segmento no es más que una memoria de 64K; la puedes considerar como una memoria aparte.

Físicamente un segmento es un cartucho con un banco de memoria cableada como la ROM del BASIC del MSX y su sistema de RAM.

Para fijar qué segmento de página se emplea, se encuentra el registro selector, en el cual habrás de introducir un valor. Lo veremos más adelante.

Generalmente los MSX tienen esta configuración de memoria:

PAGINA	SEGMENTO DEL SISTEMA	SEGMENTOS 1, 2 ó 3 CARTUCHO A CONECTAR
Página 3	RAM de 16K para los MSX de 16K	
Página 2	RAM de 16K para los MSX de 32K	ROM de 8 ó 16K para programas de juegos. Extensión de 16K para la RAM de MSX de 16K.
Página 1	ROM del BASIC del MSX	Empleada para una extensión del BASIC, ROM con el sistema operativo de diskette o para otros lenguajes.
Página 0	ROM de BASIC del MSX (BIOS)	

Cartucho

Todos los MSX tienen al menos una ranura en la que podrás colocar tu cartucho, que tienen varias funciones. Aquí tienes una lista con algunas de ellas:

1. Extensión de la RAM. Generalmente para convertir las máquinas de 16K a 32K.
2. ROM con programas de juegos. El cartucho contendrá código máquina o un programa en BASIC.
3. ROM de expansión del BASIC. Conteniendo rutinas que amplían el BASIC del MSX. La instrucción CALL es la vía de acceso a estas rutinas. Algunos cartuchos tienen su propia RAM declarada como área de trabajo.
4. Cartucho de E/S. Para controlar una unidad de *floppy disc*, interfaz de impresora o lápiz óptico. Suele tener su propia ROM para las operaciones de E/S.

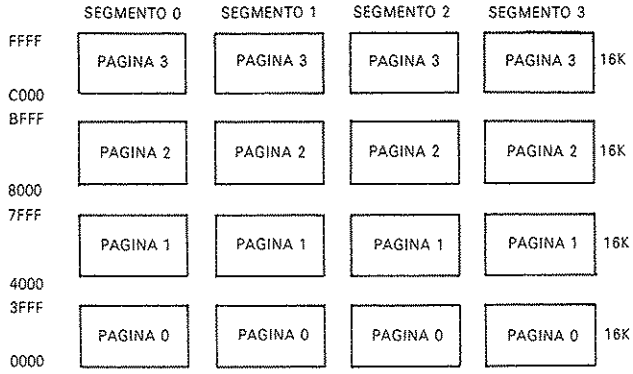
Puedes conectar cualquiera de estos cartuchos a cualquier ranura. El MSX tiene un mecanismo selector de cartucho, de modo que siempre sabrá a qué cartucho acceder.

Organización de los segmentos básicos

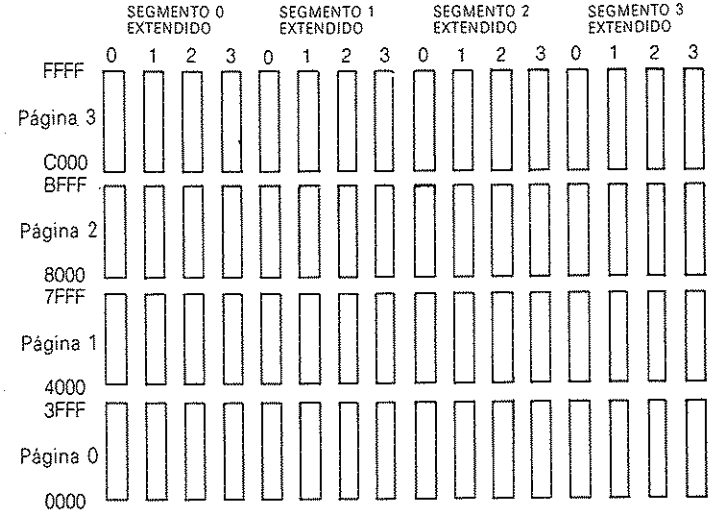
El MSX puede tener hasta cuatro segmentos; el segmento 0 es el que contiene la ROM con el BASIC del MSX (se le denomina segmento del sistema). A cada uno de estos cuatro segmentos se le puede conectar otros cuatro, teniendo así un total de dieciséis segmentos. Si cada uno de ellos tiene 64K de RAM, nos dan como resultado una memoria de 1M. Es la máxima memoria RAM que puede

llegar a direccionar el MSX. Memoria que no te es posible acceder desde el BASIC, ya que necesitas el MSX-DOS o programa en código máquina que te posibilite acceder a más de 64K de memoria.

Organización de los segmentos



Configuración de los segmentos de extensión



Selector de segmento

Ya hemos dicho que el MSX puede tener varios segmentos, pero mientras no los controlemos éstos pueden interferir entre ellos.

Para determinar qué segmento se tomará para cada página, el MSX tiene un mecanismo especial de selección de registros.

En un determinado instante la CPU puede direccionar un máximo de 64K; o sea, cuatro páginas de RAM. Cada una de estas páginas puede ser de un segmento distinto. Las páginas de los segmentos se seleccionan mediante el puerto de salida A (8255 PPI) de 8 bits. (Véase el circuito selector de segmento al pie de página.)

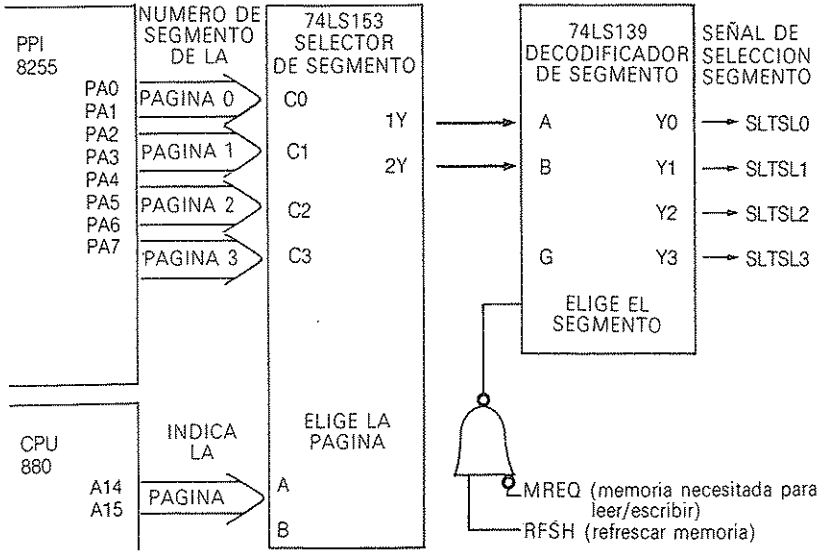
PA0 y PA1 tienen el número de segmento para la página 0.
PA2 y PA3 tienen el número de segmento para la página 1.
PA4 y PA5 tienen el número de segmento para la página 2.
PA6 y PA7 tienen el número de segmento para la página 3.

El PPI envía una señal al selector de segmento (74LS153). Este *chip* también recibe señales de la CPU (Z80) que le indican de qué página se está leyendo o escribiendo.

señal A15	señal A14	página de la que lee o escribe la CPU
0	0	página 0
0	1	página 1
1	0	página 2
1	1	página 3

Entonces se pasa el número de segmento seleccionado al decodificador 2 a 4 (74LS139), que envía la señal de selección de segmento, SLTSL, al segmento correspondiente de las cuatro páginas.

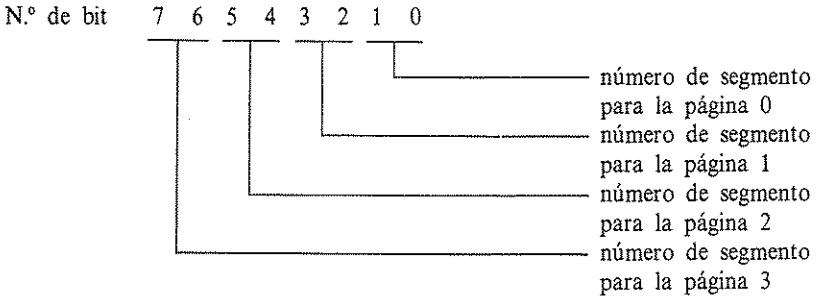
DIAGRAMA DEL CIRCUITO SELECTOR DE SEGMENTO



Cómo seleccionar y activar el segmento

Para seleccionar el segmento correspondiente a cada página por programa has de introducir un valor apropiado en la dirección &HA8, que es el puerto de salida A del PPI.

El valor a escribir es un número de 8 bits.



Ejemplo:

Supón que quieres las páginas 0, 1 y 3 del segmento 0 y la página 2 del segmento 1.

N.º de bit 7 6 5 4 3 2 1 0
0 0 0 1 0 0 0 0 = 16

Deberás escribir un 16 en la posición &HA8. Aunque la mejor manera de hacer esta operación es mediante BIOS, llamando a ENASLT (&H0024) desde el código máquina.

Segmento de extensión

El MSX puede tener hasta cuatro segmentos, del 0 al 3. No todas las máquinas los tienen, aunque te ofrecen la posibilidad de conectarlos mediante los cartuchos. El número máximo de segmentos es de 16. Si cada uno de ellos es una RAM de 64K, el MSX puede tratar una memoria de 1M bytes.

Los segmentos de extensión no pueden tener, a su vez, extensiones; estos segmentos, cartuchos conectados, son extensiones de segmentos básicos y, por ello, sus cartuchos no llevan clavijas para conectar otros cartuchos.

Para seleccionar un segmento de extensión debemos seleccionar previamente el segmento básico al que está conectado, a través del puerto de salida A del PPI 8255. El registro de selección de segmento está en la posición &HFFFF del cartucho de extensión, e indica si la página seleccionada está a uno o no.

Para saber si un segmento básico tiene conectado algún segmento de extensión, usa POKE &HFFFF, &H0F. Ahora, PRINT PEEK (&HFFFF). Si el resultado es el complemento de lo que escribiste antes, entonces dicho segmento básico tiene conectado un segmento de extensión.

Buffer del segmento de extensión

Los segmentos de extensión necesitan un *buffer*, ya que el cartucho con dicho segmento emplea un *bus* de dos direcciones.

El *buffer* cambia el sentido de transmisión según se lea o escriba. La señal de control BUSDIR es la que controla la dirección en que debe trabajar el *bus*.

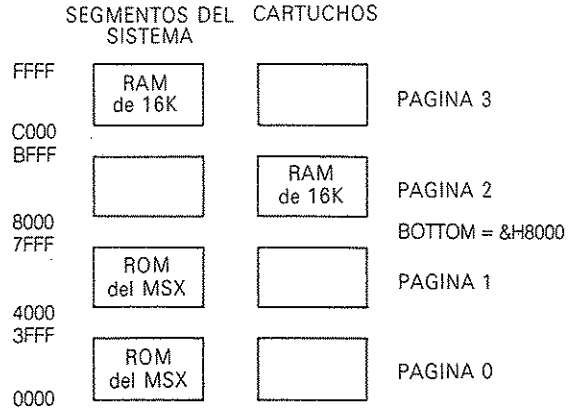
Las unidades que envían señales a la CPU, como los cartuchos de E/S, han de enviar la señal BUSDIR para cambiar el sentido de transmisión del *buffer* del segmento extendido con la CPU. Sin embargo, aquellos cartuchos que tengan sólo RAM o ROM no manejarán esta señal.

Procedimiento de búsqueda de la RAM

Cuando conectas tu MSX lo primero que hace es localizar todos los segmentos para configurar la RAM del sistema:

1. Primero busca la RAM de la página 2, de la dirección &H8000 hasta la &HBFFF, y activa el segmento con mayor capacidad de RAM en esa página. Si hay más de un segmento con este máximo, se tomará el que tenga menor número de segmento.
2. Hace lo mismo con la página 3, de la dirección &HC000 a la &HFFFF, y toma el segmento con menor número y mayor capacidad de RAM.
3. Por último, analiza si la RAM es continua desde la dirección &H8000 a la &HFFFF, y asigna a la variable del sistema BOTTOM (&HFC48) la dirección de la posición con menor dirección disponible en la RAM.

ORDENADOR DE 16K AMPLIADO A 32K



Cartuchos de ROM con programas

Procedimiento de búsqueda de la ROM

Después de seleccionar la RAM del sistema, el MSX localiza los cartuchos de ROM desde la dirección &H4000 a la &HBFFF, páginas 1 y 2. Analiza el área ID, al comienzo de cada página, desde el segmento 0 hasta el 3, además de los segmentos de extensión.

El área ID se encuentra al comienzo de cada página y tiene el siguiente formato:

+&H00	ID'AB'	ID...	Los dos bytes 'AB' indican que hay cartuchos de ROM.
+&H02	INI	INI...	Contiene la dirección de la rutina de inicialización de dicho cartucho.
+&H04	INSTRUC	INSTRUCCION...	Contiene la dirección de la rutina de tratamiento de las instrucciones ampliadas.
+&H06	UNIDAD	UNIDAD...	Contiene la dirección de la rutina de tratamiento de los periféricos ampliados.
+&H08	TEXTO	TEXTO...	Contiene la dirección de comienzo del programa BASIC en el cartucho.
+&H0A +&H10	reservado		

Nota: ID, INI, INSTRUCCION, UNIDAD y TEXTO contendrían ceros si no fuesen aplicables.

El BASIC del MSX realiza las siguientes operaciones para el procedimiento de localización del cartucho:

1. Analiza el contenido de ID y determina el tipo de rutina que contiene. Entonces pasa la información tomada al área de trabajo correspondiente.
2. Ejecuta la rutina INI, si la hubiere.
3. También ejecuta el programa BASIC, si éste existiese.

Nota: No se ejecutan ni INSTRUCCION ni UNIDAD, ya que éstas sólo se ejecutarán cuando necesites alguna instrucción ampliada o un periférico.

INI: Rutina de inicialización

El contenido de INI es la dirección de la rutina de inicialización para dicho cartucho en concreto. Generalmente inicializa alguna unidad de E/S y fija el valor

de un vector (véase el tema de los vectores), o reserva un área de trabajo para el programa del cartucho.

La rutina de inicialización varía el contenido de todos los registros, excepto el puntero de pila [SP]. Cede el control al BASIC al encontrar una instrucción RET del Z80.

Esta rutina no tiene por qué ser de inicialización; puede ser un programa en código máquina que interesa ejecutar según conectes el ordenador. Puede ser incluso un programa de juegos.

De no haber rutina de inicialización, el contenido de INI serán ceros.

TEXTO: Programa BASIC

Un cartucho de ROM no tiene por qué llevar sus programas escritos en código máquina: también pueden estar en BASIC. En TEXTO se halla la dirección de comienzo del programa BASIC que contiene el cartucho.

Cuando programes un cartucho en BASIC has de tener en cuenta lo siguiente:

1. Si tienes más de un cartucho con programas en BASIC conectado a tu ordenador, éste sólo ejecutará el que tenga menor número de segmento.
2. Los programas en BASIC de los cartuchos se almacenan en *tokens*.
3. Los cartuchos de este tipo han de estar asociados solamente a la página 2 (&H8000-&HBFFF). Por tanto, su capacidad máxima es de 16K.
4. La página 2 de cualquier otro segmento con RAM se desactivará y no se podrá emplear mientras se ejecute el cartucho con el programa en BASIC.
5. La dirección a la que apunta TEXTO ha de contener un cero.
6. Los números de línea de las instrucciones de salto, GOTO, GOSUB, etc., se constituirán por punteros para acelerar la ejecución.

En caso de que el cartucho no tenga un programa BASIC, TEXT estará a cero.

INSTRUCCION: Rutina de ampliación de instrucciones

Mediante la instrucción CALL puedes llamar a ciertas instrucciones ampliadas que no estén en la ROM del BASIC del MSX. En INSTRUCCION está la dirección de comienzo de la rutina que ejecuta la primera instrucción ampliada del BASIC. El cartucho ha de estar en la página 1 de cualquier segmento, excepto en el segmento del sistema, que es el que contiene el BASIC.

La sintaxis de las instrucciones ampliadas es:

```
CALL <nombre-instrucción>
CALL <nombre-instrucción> (argumentos)
```

La instrucción CALL se puede sustituir por un signo '_'.

Cuando el BASIC encuentra una instrucción CALL, almacena el nombre de la instrucción extendida en el área de trabajo del sistema llamada PROCNM. Esta área comienza en la dirección &HFD89 y tiene 16 bytes de longitud. El nombre de la instrucción extendida se guarda en PROCNM seguido de un cero, por lo que las instrucciones no podrán tener más de quince caracteres.

Antes de ejecutar la instrucción ampliada, el puntero de texto, o sea, el registro [HL], contendrá la dirección que sigue a INSTRUCCION.

El procedimiento de INSTRUCCION analiza el contenido de PROCNM y ejecuta la rutina que corresponda a dicho nombre de instrucción.

Después de ejecutar la instrucción ampliada, el *flag* de acarreo se pondrá a 0 y el puntero de texto HL apuntará a la siguiente instrucción a ejecutar.

Si la instrucción ampliada almacenada en PROCNM no tuviese una rutina para ejecutarla, entonces el puntero de texto y el *flag* de acarreo quedarían como estaban y se enviará al BASIC un error sintáctico, mostrándote el mensaje *Syntax error* en pantalla.

En caso de que el cartucho no tuviese ninguna rutina de ejecución de instrucciones ampliadas, el campo INSTRUCCION estará a 0.

UNIDAD: Rutina de tratamiento de extensiones de periféricos

El MSX te permite que le conectes cartuchos de extensión de E/S. El campo UNIDAD contendrá la dirección de comienzo de la rutina de tratamiento del periférico.

Recuerda estos puntos sobre las rutinas de UNIDAD:

1. El cartucho ha de estar conectado a la página 1 (&H4000-&HBFFF).
2. A un cartucho (16K) se le pueden conectar hasta cuatro unidades lógicas.
3. El nombre de la unidad se almacena en PROCNM, igual que se hacía en INSTRUCCION. PROCNM comienza en la dirección &HFD89 y tiene una longitud de 16 bytes. El nombre de la unidad va seguido de un 0 al guardarse en PROCNM; por tanto, la longitud máxima del nombre de una unidad es de dieciséis caracteres.
4. Cuando en una instrucción OPEN, u otras, el BASIC encuentra un nombre de unidad que no sea reconocida por la ROM del MSX, entonces hace una llamada a la unidad, metiendo &HFF en el acumulador. En caso de no existir una rutina que trate dicha unidad en el cartucho pone a 1 el *flag* de acarreo. Si ésta existe, el contenido de ID (valor de 0 a 3) se lleva al acumulador y el *flag* de acarreo se pone a 0. La rutina de dicha unidad puede trabajar con todos los registros.
5. Las operaciones de E/S se hacen realmente cuando se encuentre a UNIDAD con uno de estos valores en el acumulador:

   0 Abrir (OPEN).
   1 Cerrar (CLOSE).
   4 E/S aleatoria.
   6 Salida secuencial.
   8 Entrada secuencial.
  10 Función LOC.
  12 Función LOF.
  14 Función EOF.
  16 Función FPOS.
  18 Carácter de retroceso.

La variable del sistema UNIDAD ha de tener el contenido del indicador de unidad ID (0-3).

Si el cartucho no tuviera ninguna rutina de tratamiento de periféricos, entonces UNIDAD contendrá cèros.

Descripción de las variables del sistema relacionadas con el mecanismo de los segmentos

Estado de los segmentos

EXPTBL: Indica los segmentos ampliados.
Está en la posición &HFCC1 y tiene cuatro bytes de longitud.

EXPTBL
- &HFCC1 segmento 0.
- &HFCC2 segmento 1.
- &HFCC3 segmento 2.
- &HFCC4 segmento 3.
  - &H80 indica segmento ampliado.
  - &H00 no hay ampliación.

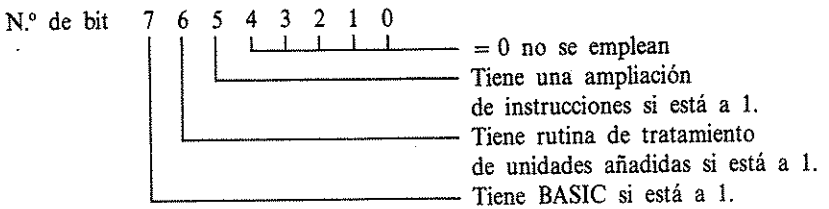
SLTTBL: Indica el valor que se está sacando al registro selector de segmento de extensión. Sólo es válida en el caso de que el correspondiente EXPTBL tenga &H80; o sea, cuando se ha ampliado dicho segmento.
Está en la posición &HFFC5 y tiene cuatro bytes de longitud.

SLTTBL
- &HFFC5 segmento 0.
- &HFFC6 segmento 1.
- &HFFC7 segmento 2.
- &HFFC8 segmento 3.

Estado de las páginas

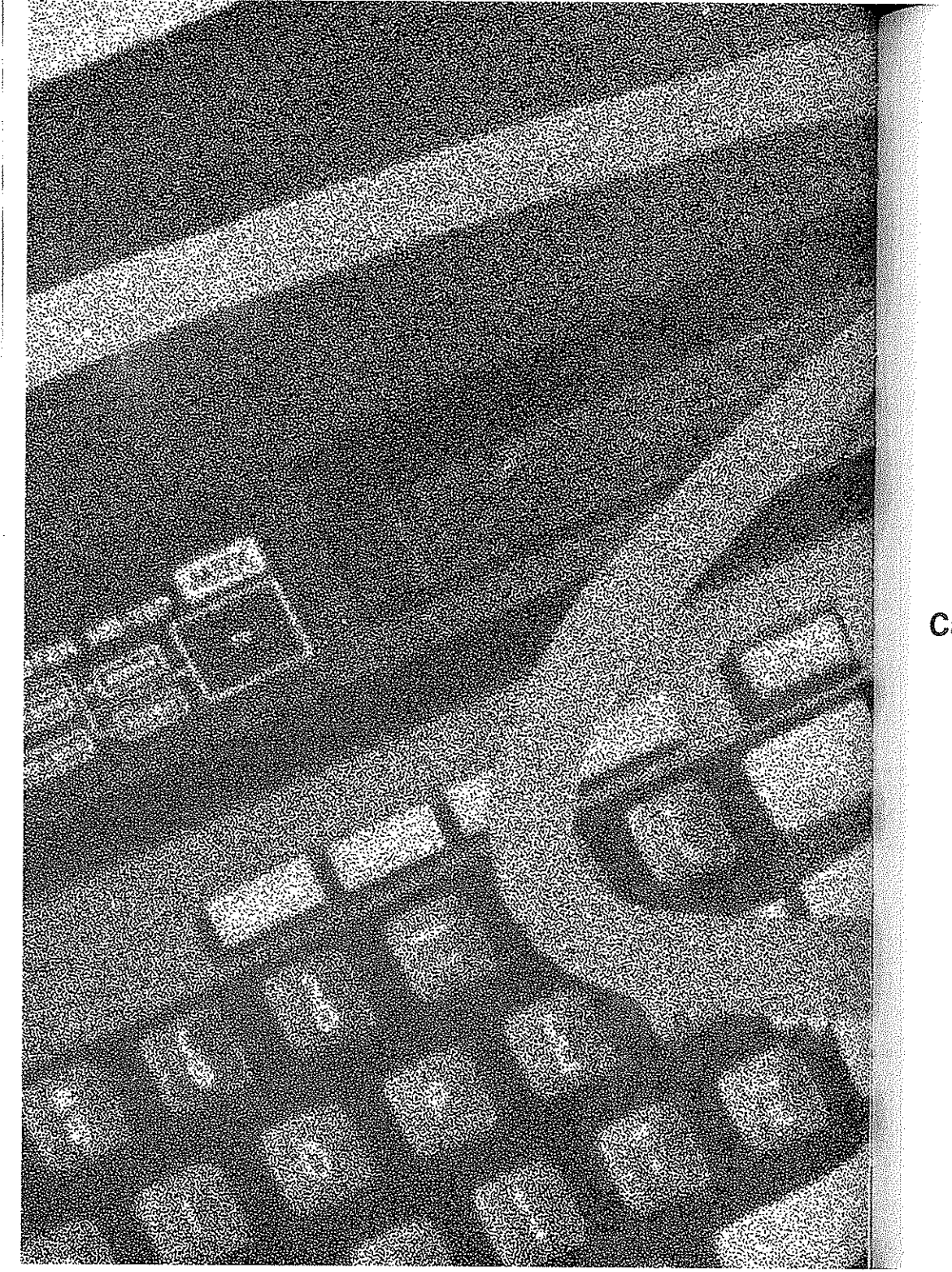
SLTATR: Indica el contenido de todas las páginas.
Está en la posición &HFCC9 y tiene 64 bytes de longitud.

SLTATR
- &HFCC9 segmento básico 0 segmento ampliado 0 página 0.
- &HFCCA segmento básico 0 segmento ampliado 0 página 1.
- ... ...
- ... ...
- &HFD07 segmento básico 3 segmento ampliado 3 página 2.
- &HFD08 segmento básico 3 segmento ampliado 3 página 3.



SLTWRK: Area de trabajo de cada página. Su uso depende del contenido de la página, aunque cada página tiene dos bytes para indicar un área de trabajo, independientemente de su contenido.
Está en la posición &HFD09 y tiene 128 bytes de longitud, 2 por página.

SLTWRK
- &HFD09 segmento básico 0 segmento ampliado 0 página 0.
- &HFD0A segmento básico 0 segmento ampliado 0 página 0.
- &HFD0B segmento básico 0 segmento ampliado 0 página 1.
- &HFD0C segmento básico 0 segmento ampliado 0 página 1.
- ... ...
- ... ...
- &HFD85 segmento básico 3 segmento ampliado 3 página 2.
- &HFD86 segmento básico 3 segmento ampliado 3 página 2.
- &HFD87 segmento básico 3 segmento ampliado 3 página 3.
- &HFD88 segmento básico 3 segmento ampliado 3 página 3.



51
Periféricos

Cassette

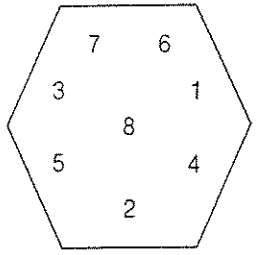
Estas son todas las órdenes y funciones que tiene el MSX relacionadas con el cassette. (En la *Guía de referencia del BASIC* te damos una explicación más detallada de cada una de ellas.)

CLOAD
CSAVE
MOTOR
SAVE
BSAVE
BLOAD

Conexión: Conector DIN tipo 45326 con ocho clavijas.

Tabla de señales

N.º	SEÑAL	SENTIDO
1	GND	...
2	GND	...
3	GND	...
4	CMT OUT	SALIDA
5	CMT IN	ENTRADA
6	REM+	SALIDA
7	REM−	SALIDA
8	GND	...



Impresora

Si tu ordenador MSX tiene interfaz de impresora puedes utilizar cualquier tipo de impresora que tenga un interfaz Centronics. Si tu ordenador no tiene interfaz de impresora debes comprar un cartucho que lo incorpore y que, conectado a la ranura, te permita emplearla.

Algunas impresoras están equipadas con los caracteres estándar del MSX. Hay varios fabricantes que disponen de ellas; te proporcionan la ventaja de poder imprimir todo el juego de caracteres gráficos de tu MSX. Una impresora que no sea MSX no podrá imprimir ningún carácter de este juego; en su lugar imprimirá un espacio por cada carácter gráfico.

Si utilizas una impresora MSX has de indicárselo al ordenador mediante la instrucción SCREEN:

```
SCREEN,,,,0  impresora MSX
SCREEN,,,,1  impresora no MSX
```

LPRINT
LPRINT USING

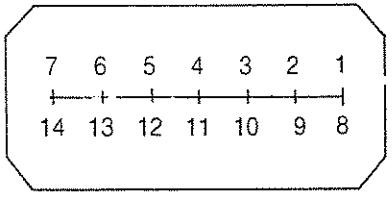
Para escribir en la impresora debes utilizar las instrucciones LPRINT y LPRINT USING, en lugar de PRINT y PRINT USING. Su sintaxis es la misma, excepto que no mostrarás nada por pantalla. Si quieres escribir en pantalla y por impresora emplea ambas instrucciones a la vez (PRINT y LPRINT).

La instrucción LLIST lista el programa por impresora. Su sintaxis es la misma que la del comando LIST.

La función LPOS devuelve la posición de la cabeza de la impresora y es análoga a la función POS, que proporciona la posición del cursor en las pantallas de texto.

Conector: tipo Centronics, 14 clavijas Aphenol.

N.º PUNTO	NOM.-SEÑAL
1	PSTB
2	PDB0
3	PDB1
4	PDB2
5	PDB3
6	PDB4
7	PDB5
8	PDB6
9	PDB7
10	NC (sin función)
11	BUSY
12	NC (sin función)
13	NC (sin función)
14	GND



Cuando te compres la impresora te darán más detalles en su manual. Este incluye programas de demostración, escritos en el BASIC de Microsoft, compatible con el MSX.

Puertos del *joystick*

La mayoría de los ordenadores MSX tienen dos puertos de *joystick*, aunque algunos sólo tienen uno: el BASIC permite un máximo de dos. Aparte de los *joystick*, puedes conectar otros dispositivos como la pizarra electrónica y los *paddles*.

Conector: AMP de 9 clavijas.

Joystick

Se emplean principalmente para los juegos. Si tu ordenador tiene dos puertos para el *joystick* y tú sólo uno, asegúrate de conectarlo al puerto con la marca JOYSTICK 1, ya que la mayoría de los programas comerciales asumen que está conectado a este puerto.

Para saber el estado del *joystick* desde el BASIC emplea las órdenes STICK y STRIG. STICK nos da la dirección de movimiento del *joystick*, y STRIG te dice el estado del disparador.

La instrucción ON STRIG GOSUB te permite las interrupciones del disparador. (Véase el tema de interrupciones y sucesos del BASIC.)

Paddle

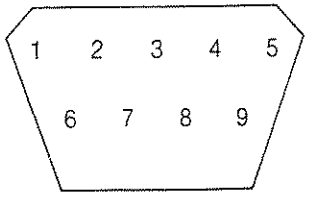
Puedes conectar hasta doce *paddles*, seis a cada puerto del *joystick*. El estado del *paddle* se sabe mediante la función PDL. (Véase PDL.)

Pizarra electrónica

Puedes conectar una de estas pizarras por cada puerto del *joystick*.

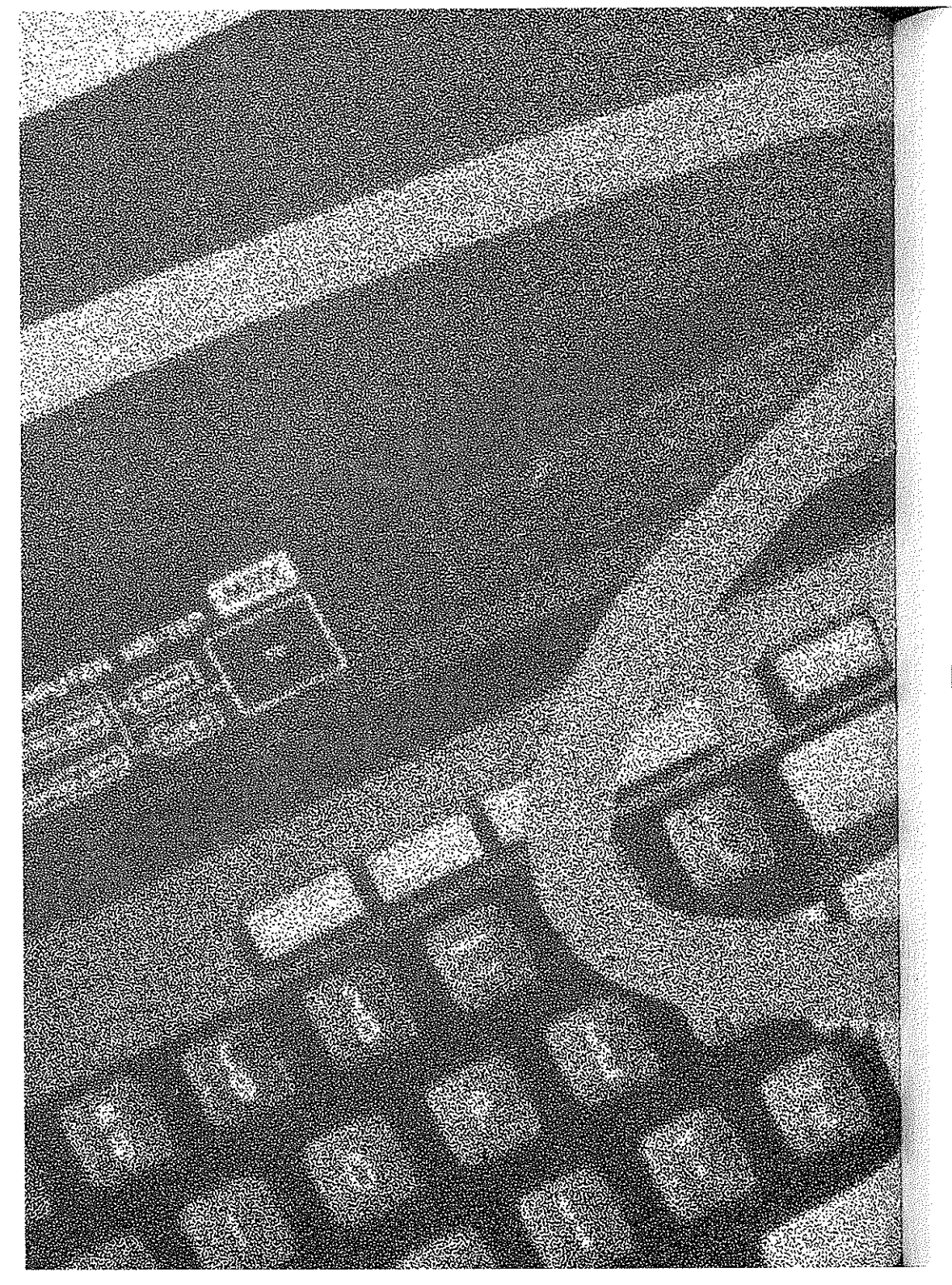
PAD te da el estado de la pizarra. (Véase PAD en la *Guía de referencia del BASIC.*)

	SEÑAL	SENTIDO
1	FWD	ENTRADA
2	BACK	ENTRADA
3	LEFT	ENTRADA
4	RIGHT	ENTRADA
5	+5V	...
6	TRG 1	ENTRADA
7	TRG 2	ENTRADA
8	OUTPUT	SALIDA
9	GND	...



PARTE TERCERA

SECCION DE CONSULTA



52

Instrucciones del BASIC

Formato de las palabras claves

En este tema se incluyen todas las instrucciones del BASIC del MSX, y todo lo que necesitas saber sobre ellas.

La explicación de cada instrucción está estructurada de tal modo que te será más fácil entenderlas. Dicha descripción está dividida así:

INSTRUCCION

Seguida, en algunos casos, de su derivación inglesa.

DESCRIPCION

Explica, sencillamente, qué función realiza la instrucción.

SINTAXIS

Enumera todas las formas sintácticas de la instrucción, con su correspondiente explicación, así como las diferencias con otras sintaxis de dicha instrucción.

Usaremos los siguientes símbolos:

<const-num>	Es un número, tal cual; por ejemplo, 100 ó 1.414.
<var-num>	Es una variable numérica como X o SUMATORIO.
<exp-num>	Es una expresión cuyo resultado es un número como 2*PI*R*COS(Y). <const-num> y <var-num> se pueden usar como expresiones de resultado numérico, y se podrán emplear en donde sea necesaria una expresión numérica.
<numérico>	Es cualquiera de las constantes, variables o expresiones antes mencionadas.
<const-car>	Es una cadena de caracteres entre comillas; por ejemplo: "ORDENADOR MSX".
<var-car>	Es una variable de caracteres como B$ o PALABRA$.
<exp-car>	Es una expresión cuyo resultado es una cadena de caracteres, como: B$+"AEROPUERTO","4" o STR$(1234).
<variable>	Puede ser <var-car>, o bien <var-num>.
<condición>	Es una condición de respuesta VERDADERO O FALSO, como, por ejemplo, A<0 o BS="AEROPUERTO".
<comando>	Puede ser cualquier comando BASIC de un determinado grupo de ellos.
<línea>	Es un número de línea.
<nombre>	Puede ser el nombre de un programa o bien una función.

EJEMPLOS

En esta parte se incluyen varios ejemplos ilustrativos y algún programa corto que te mostrará cómo debe emplearse la instrucción. También daremos una breve explicación del ejemplo.

En los programas cortos usaremos los siguientes formatos:

```
Todo lo que este encerrado
en este cuadrado es un programa
que podras ejecutar (RUN) en tu ordenador
```



```
En este otro cuadrado tendras
los resultados de la ejecucion.
A veces te daremos una explicacion complementaria
en este mismo cuadrado y junto a la linea.
```

PUNTOS A RECORDAR

Aquí se detalla lo que hace y lo que no hace la instrucción. Te daremos de forma razonada las descripciones suplementarias e indicaciones de la instrucción.

PRECAUCIONES

Te muestra, para que puedas depurar con más facilidad tus programas, posibles causas de errores en la instrucción.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Esta sección te enumera todas las instrucciones que se suelen utilizar en combinación con la instrucción descrita. También te indicamos en qué parte del libro se menciona dicha instrucción.

ABS Valor ABSoluto *(ABSolute value)*

DESCRIPCION

Esta función devuelve el valor absoluto, que es la distancia entre el número del paréntesis y el cero; esto es, los números negativos se transforman en positivos, y los positivos se mantienen. Por ejemplo, el valor absoluto de −2,6 es 2,6.

Se emplea también para calcular la diferencia positiva entre dos números.

SINTAXIS

ABS(<const-num>)
ABS(<var-num>)
ABS(<exp-num->)

EJEMPLO

```
10 PRINT "EL VALOR ABSOLUTO DE -1000 ES ";
   ABS(-1000)
20 A=1984
30 B=1960
40 PRINT ABS(B-A)
```

PUNTOS A RECORDAR

ABS es una función con argumento y resultado de tipo numérico. El valor ABS de una variable no tratada es cero.

PRECAUCIONES

La mezcla de una función ABS con una cadena de caracteres dará un error en el tipo datos *(Type mismatch error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SGN
Parte primera, tema 15: "Funciones".

AND Operador Y lógico

DESCRIPCION

AND es uno de los operadores lógicos que emplean a formar parte de las condiciones múltiples y emplean el álgebra de Boole, que consiste una parte

El operador AND suele usarse dentro de la instrucción IF ... THEN ... ELSE para preguntar por más de una condición a la vez. Esta se separa de la otra instrucción. Por ejemplo:

```
IF A=5 AND B$="CORCEL" THEN PRINT "¡UHHH!" ELSE PRINT
"VALE"
```

La operación AND (Y-lógico) la podemos representar mediante una tabla de verdad:

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Por tanto, 100 AND 50 lo podremos calcular como:

```
X  100   0000000001100100
Y   50   0000000000110010
--------------------------
   32    0000000000100000
```

SINTAXIS

IF <condición> AND <condición> ... THEN ... ELSE
<var-num> = <const-num> AND <const-num>
<var-num> = <var-num> AND <exp-num>
u otras combinaciones numéricas.

EJEMPLOS

Ejemplo 1): En la condición de la instrucción IF THEN ELSE:

```
10 T=12 : HAMBRE$="MUCHA"
20 INPUT "QUIERES COMER (S/N)";R$
30 IF T=12 AND HAMBRE$="MUCHA" AND R$="S" THEN PRI
NT "TOMATE UN BOCATA." : END
40 PRINT "DE ACUERDO TE DAREMOS ALGO MAS TARDE."
```

AND Operador Y lógico

DESCRIPCION

AND es uno de los operadores lógicos que entran a formar parte de las condiciones múltiples y emplean el álgebra de Boole, que constituye una parte esencial dentro de la informática; en el tema del álgebra de Boole encontrarás más detalles.

El operador AND suele usarse dentro de la instrucción IF ... THEN ... ELSE para preguntar por más de una condición antes de ejecutar una u otra instrucción. Por ejemplo:

IF A=5 AND B$="CORCEL" THEN PRINT "¡IIHHH!" ELSE PRINT "VALE"

La operación AND (Y-lógico) la podemos representar mediante una tabla de verdad:

AND	X	Y	X AND Y
	0	0	0
	0	1	0
	1	0	0
	1	1	1

Por tanto, 100 AND 50 lo podremos calcular como:

X	100	0000000001100100
Y	50	0000000000110010
	32	0000000000100000

SINTAXIS

IF <condición> AND <condición> ... THEN ... ELSE
<var-num> = <const-num> AND <const-num>
<var-num> = <var-num> AND <exp-num>
u otras combinaciones numéricas.

EJEMPLOS

Ejemplo 1): En la condición de la instrucción IF THEN ELSE:

```
10 T=12 : HAMBRE$="MUCHA"
20 INPUT "QUIERES COMER (S/N)";R$
30 IF T=12 AND HAMBRE$="MUCHA" AND R$="S" THEN PRI
NT "TOMATE UN BOCATA." : END
40 PRINT "DE ACUERDO TE DAREMOS ALGO MAS TARDE."
```


Ejecuta (RUN) este programa dos veces y observa lo que ocurre si respondes no (N) o sí (S).

```
RUN
QUIERES COMER (S/N)? S
TOMATE UN BOCATA.
OK
RUN
QUIERES COMER (S/N)? N
DE ACUERDO TE DAREMOS ALGO MAS TARDE.
```

Ejemplo 2): En el álgebra de Boole.

```
10 A%=185 : B%=85
20 C%=A% AND B%
30 PRINT A%;" AND ";B%;" = ";C%
40 PRINT "SI LO MULTIPLICAS POR 100 TENDRAS";100*(
A% AND B%)
```

```
 185  AND  85  =  17
SI LO MULTIPLICAS POR 100 TENDRAS 1700
```

PUNTOS A RECORDAR

La operación AND(Y-lógico) se puede emplear para conocer un determinado bit. Por ejemplo, si Y = 2", siendo *n* el bit a conocer, un número entre 0 y 7; si X AND Y = Y, entonces el bit a conocer era 1 (verdadero); en cambio, si X AND Y = 0 el bit a conocer era 0 (falso).

Analizamos el quinto bit (n = 5) de X = 117.

```
Y = 2^5 = 32
      X  117     0000000001110101
AND   Y   32     0000000000100000   X AND Y = Y
          32     0000000000100000
```

luego el quinto bit de X = 117 es 1 (verdadero).

Los operandos numéricos en las operaciones booleanas han de estar comprendidos entre −32768 y 32767; los números reales que entren en dichas operaciones se truncan para convertirse en números enteros.

PRECAUCIONES

Cuando alguno de los operandos se sale de rango se producirá un error de desbordamiento *(Overflow error)*. En cambio, si algún operando es una cadena de caracteres, tendremos un error en el tipo de datos *(Type mismatch error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

IF
THEN
ELSE
OR
EQV
XOR
NOT
IMP

Parte primera, tema 6: "Las condiciones".
Parte segunda, tema 34: "Algebra de Boole (I): operadores lógicos".
Parte segunda, tema 35: "Algebra de Boole (II): la instrucción IF/THEN/ELSE".

ASC Código ASCII

DESCRIPCION

A todos los caracteres les corresponde un código numérico, llamado código ASCII *(American Standard Code for Information Interchange)*. El ordenador convierte todas las cadenas de caracteres en números, ya que trabaja mejor con números que con caracteres. La aparición del código ASCII tiene el objetivo de que todos los ordenadores que trabajen con él tengan los mismos códigos de caracteres alfanuméricos, con lo cual se facilita la comunicación entre ordenadores.

A veces resulta necesario conocer el código ASCII de algún carácter; para ello, la función ASC nos devuelve el código de un carácter o del primero de una cadena de caracteres.

SINTAXIS

ASC("<cad-car>")	Nos devuelve el código del primer carácter de la cadena. La cadena puede tener más de un carácter.
ASC(<var-cad>)	Nos devuelve el código del primer carácter que contiene la variable de caracteres.

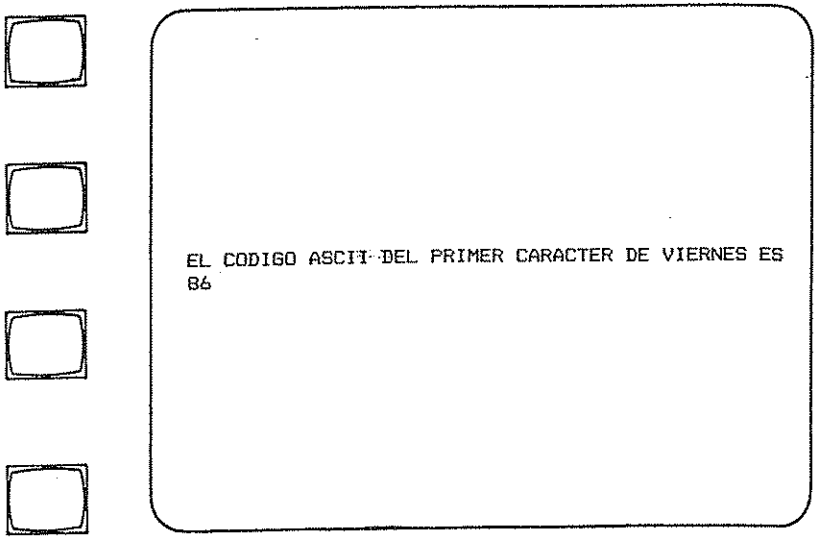
EJEMPLO

```
PRINT ASC("A")
```

te dará por pantalla el código ASCII del carácter "A", que es 65.

```
10 A$="VIERNES"
20 B=ASC(A$)
30 PRINT "EL CODIGO ASCII DEL PRIMER CARACTER DE "
;A$;" ES";B
```

El programa anterior te dará el siguiente resultado:



PUNTOS A RECORDAR

Todos los caracteres, incluyendo los de control como <return> y los caracteres gráficos, tienen un único código ASCII, y éste puede ser calculado mediante la función ASC.

Solamente el primer carácter de una cadena es significativo para la función ASC; los caracteres restantes los ignora.

La función inversa, que es la de devolver el carácter correspondiente a un número perteneciente al rango que delimita el código ASCII, la efectúa la función CHR$, la cual devuelve como máximo un carácter.

PRECAUCIONES

Si a una variable de tipo cadena de caracteres, que no contiene nada —cadena nula—, le aplicamos la función ASC, nos dará un error de llamada a una función ilegal *(Illegal function call)*.

Por ejemplo: 10 PRINT ASC("") nos dará un error de llamada a una función ilegal en la línea 10.

En cambio se producirá un error en el tipo de datos asignado a la función *(Type mismatch)* si el argumento es numérico.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CHR$

(Véanse los apéndices para consultar la tabla de códigos ASCII.)

Parte primera, tema 20: "El código ASCII".

ATN ArcoTaNgente

DESCRIPCION

Esta función devuelve el arcotangente de un número, expresando el valor en radianes, comprendido entre $-\pi/2$ y $\pi/2$.

SINTAXIS

ATN(<const-num>)
ATN(<var-num>)
ATN(<exp-num>)

El valor del arcotangente que devuelve la función ATN se encuentra en formato de doble precisión.

EJEMPLOS

```
PRINT ATN (1.5)
   .98279372324731
PRINT ATN (3/6)
   .46364760900081
```

PI = 4*ATN (1)... 4*ATN (1) calcula el valor de PI (π) con catorce cifras decimales.

PUNTOS A RECORDAR

La función ATN devuelve siempre un número en doble precisión.

El argumento puede ser de cualquier tipo numérico.

PRECAUCIONES

Al ser una función trigonométrica no debe ser mezclada con cadenas de caracteres; de ocurrir esto, nos dará un error en el tipo de datos empleado *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

TAN
SIN
COS

Parte primera, tema 19: "Funciones matemáticas".

AUTO — Numeración AUTOmática de líneas

DESCRIPCION

Los programas en BASIC han de llevar una numeración en cada línea del programa. Cuando se están escribiendo programas largos, la instrucción AUTO nos numerará la línea siguiente de modo automático cada vez que pulsemos la tecla <return> al finalizar una línea.

Con ello se facilita el trabajo del programador. Los números de línea se irán incrementando en una cantidad constante.

SINTAXIS

AUTO	Numera las líneas desde 10, incrementándose de 10 en 10.
AUTO <const-num>	Numera las líneas desde la especificada, incrementándose de 10 en 10.
AUTO <const-num>,<const-num>	Numera las líneas desde la especificada, con el incremento especificado en la segunda constante numérica.
AUTO,<const-num>	Numera las líneas desde la 0 con el incremento especificado.
AUTO <const-num>	Numera las líneas desde la especificada, con un incremento igual al que se empleaba anteriormente.

EJEMPLOS

AUTO numera las líneas: 10, 20, 30, 40...

AUTO 1000,5 numera las líneas: 1000, 1005, 1010...

PUNTOS A RECORDAR

Cada vez que se teclee <return> el ordenador numerará la siguiente línea. Hay dos modos para salirse de la ejecución de AUTO:

1) Tecleando <CTRL> y <STOP> simultáneamente.
2) Tecleando <CTRL> y <C> simultáneamente.

Los números de línea han de estar comprendidos entre 0 y 65529, y los incrementos pueden ir de 1 a 65529. Cuando se haya llegado a la línea 65529 el ordenador dejará de dar nuevos números de línea y se pondrá en el modo directo.

Si en el programa que se está editando aparece un número de línea seguido de un "*", se nos está indicando que esa línea ya estaba editada; si no quieres cambiarla pulsa <return>; en caso contrario continúa escribiendo normalmente, ignorando el "*".

La tecla de función (F-2) está programada como "AUTO", siempre que tú no redefinas dicha función; luego, presionando esa tecla, entrarás al modo AUTO.

Los números de línea han de ser números enteros.

PRECAUCIONES

Si sucede que ni el número de línea de comienzo, ni el incremento, son números enteros, habrá un error sintáctico *(Syntax error)*.

Si no prestas atención será fácil que, estando en el modo AUTO, borres un programa, ten cuidado cuando tengas un número de línea seguido de un "*".

La instrucción AUTO puede ir incluida dentro de un programa, pero esto no es realmente muy efectivo y, además, puede darte problemas.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

RENUM

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

Parte segunda, tema 29: "Edición avanzada de programas".

BASE

DESCRIPCION

Esta variable del sistema te proporciona la dirección en la que se encuentra el puntero de cada una de las tablas de pantalla residentes en la RAM de video. BASE es una variable como otra cualquiera; por tanto, la puedes asignar valores y emplearla dentro de expresiones, pero sólo en los programas que trabajen con gráficos avanzados. BASE es un número entero de 16 bits.

SINTAXIS

BASE(<número>)

BASE(0) — Dirección apuntada en la tabla de símbolos del modo de texto 0.

BASE(1) — No se usa.

BASE(2) — Dirección apuntada en el generador de caracteres para el modo de texto 0.

BASE(3) — No se usa.

BASE(4) — No se usa.

BASE(5) — Dirección apuntada en la tabla de símbolos del modo de texto 1.

BASE(6) — Dirección apuntada en la tabla de color del modo de texto 1.

BASE(7) — Dirección apuntada en el generador de caracteres para el modo de texto 1.

BASE(8) — Dirección apuntada en la tabla de características de *sprites* del modo de texto 1.

BASE(9) — Dirección apuntada en la tabla de modelos de *sprites* del modo de texto 1.

BASE(10) — Dirección apuntada en la tabla de símbolos en el modo gráfico de alta resolución 2.

BASE(11) — Dirección apuntada en la tabla de color de alta resolución del modo gráfico de alta resolución 2.

BASE(12) — Dirección apuntada en el generador de caracteres del modo gráfico de alta resolución 2.

BASE(13) — Dirección apuntada en la tabla de características de *sprites* del modo gráfico de alta resolución 2.

BASE(14) — Dirección apuntada en la tabla de diseños de *sprites* del modo gráfico de alta resolución 2.

BASE(15) — Dirección apuntada en la tabla de símbolos del modo gráfico multicolor 3.

BASE(16) — No se usa.

BASE(17) — Dirección apuntada en el generador de caracteres del modo gráfico multicolor 3.

BASE(18) — Dirección apuntada en la tabla de características de *sprites* del modo gráfico multicolor 3.

BASE(19) — Dirección apuntada en la tabla de modelos de *sprites* del modo gráfico multicolor 3.

EJEMPLO

```
PRINT BASE(0)
0
```

PRECAUCIONES

El argumento de esta variable ha de estar comprendido entre 0 y 19, y ha de ser un número entero. En caso de no ser así, se producirá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 44: "Gráficos avanzados (V): la RAM de video".

BEEP pitido del altavoz

DESCRIPCION

La instrucción BEEP nos da exactamente el mismo sonido que si utilizamos PRINT CHR$(7), con una duración de 0,04 segundos.

SINTAXIS

```
BEEP
```

EJEMPLO

```
BEEP
```

PUNTOS A RECORDAR

La instrucción BEEP es equivalente a CHR$(7).

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SOUND
PLAY

Parte primera, tema 28: "El macrolenguaje musical".

BIN$ Cadena BINaria

DESCRIPCION

La función BIN$ nos da el valor en el sistema binario (base 2) de un argumento en sistema decimal (base 10) en forma de cadena. El argumento ha de ser un número entero comprendido entre −32768 y 65535, o también una expresión con un resultado entero comprendido en dicho rango.

SINTAXIS

```
BIN$(<const-num>)
BIN$(<var-num>)
BIN$(<exp-num>)
```

EJEMPLO

```
10 PRINT BIN$(255)
20 PRINT BIN$(9999)
```

```
RUN
11111111
10011100001111
```

PUNTOS A RECORDAR

Si el argumento es negativo el número binario equivalente estará en complemento a 2: o sea, BIN$(−n) = BIN$(65536 − n).

Para convertir un número binario a decimal ponle el prefijo &B e iguálalo a una variable decimal. Observa este ejemplo:

```
A% = &B00001111
```

PRECAUCIONES

El argumento de esta función no puede ser ni un número real ni una cadena de caracteres; de ser así se produciría un error en el tipo de datos *(Type mismatch)*.

Si el argumento es un número entero, pero fuera de rango (−32768, 65535), te causará un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Segunda parte, tema 33: "Presentación de los sistemas de numeración del MSX".

BLOAD Carga de bytes *(LOAD Bytes)*

DESCRIPCION

La instrucción BLOAD se emplea para cargar en memoria datos y programas en código máquina desde un determinado periférico; por ejemplo, el cassette. Ofrece, además, las opciones de ejecutar el programa cargado en código máquina, y la de cargar dicho programa en otras posiciones de memoria diferentes desde las que fue grabado al periférico.

BLOAD se suele emplear para autoejecutar programas en código máquina.

De momento esta instrucción sólo es válida para unidades de cassette, pero en un futuro su uso se extenderá a unidades de disco.

SINTAXIS

BLOAD "<nombre-unidad>:"	Para cargar ficheros de nombre desconocido.
BLOAD "<nombre-unidad>:<nombre>"	Para cargar ficheros en caso de que conozcamos su nombre.
BLOAD "<nombre-unidad>:<nombre>",R	De este modo se ejecuta el programa cargado en código máquina. La R viene de RUN.
BLOAD "<nombre-unidad>:<nombre>",<const-num>	Para cargar a partir de la dirección especificada en el programa más <const-num>.
BLOAD "<nombre-unidad>:<nombre>",R,<num-const>	Igual que el anterior, con autoejecución.

EJEMPLOS

```
BLOAD "CAS:JUEGO",R
BLOAD "CAS:LLEGA",&H00FF
```

PUNTOS A RECORDAR

Si se omite la dirección de carga cuando se graba el fichero (BSAVE), el ordenador toma ésta como la dirección a partir de la cual estaba el programa en memoria, habiendo sido cargado con BLOAD especificando la opción R.

Gran parte de los programas de juegos se cargan y ejecutan con este comando.

La longitud del nombre del fichero ha de ser como máximo de seis caracteres.

PRECAUCIONES

Si el fichero está distorsionado en el cassette, te producirá un error en el dispositivo de E/S *(Device I/O error)*.

No conseguirás cargar el programa deseado si has tecleado mal su nombre, cosa frecuente. Si no recuerdas bien el nombre del fichero te recomendamos que

no lo especifiques, ya que el ordenador cargará el programa en el primer código máquina que encuentre.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

BSAVE

Parte segunda, tema 39: "Grabación y carga con el cassette".

BSAVE Grabación de bytes *(SAVE Bytes)*

DESCRIPCIÓN

Graba parte del contenido de la memoria en una unidad de cassette o disco. Debes especificar las direcciones de comienzo y final. Este comando se emplea para grabar en forma de bytes datos y programas en código máquina. También puedes especificar la dirección de comienzo de la ejecución, en el caso de que luego lo cargues con BLOAD y lo ejecutes automáticamente.

SINTAXIS

BSAVE "<nombre-unidad>:<nombre>",<dir-comienzo>,<dir-final>
BSAVE "<nombre-unidad>:<nombre>",<dir-comienzo>,<dir-final>,
<dir-ejecución>

EJEMPLOS

BSAVE "CAS:JUEGO",&HA100,&HA200,&HA1FF
BSAVE "CAS:LLEGA",&HC000,&HCFFF

PUNTOS A RECORDAR

La longitud en caracteres del nombre del fichero ha de ser menor o igual que 6.

Si cargas y ejecutas el programa con la instrucción BLOAD y la opción R, y además no has especificado dirección de comienzo de la ejecución, el ordenador toma como dirección de ejecución la de comienzo del programa.

PRECAUCIONES

Sólo lograrás una buena grabación de la memoria si la unidad que utilizas está en perfectas condiciones de funcionamiento. En caso de que no puedas grabar correctamente revisa tu unidad de E/S.

Un nombre de fichero que tenga más de seis caracteres ocasionará un error de sintaxis *(Syntax error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 39: "Grabación y carga con el cassette".

CALL Llamada a una instrucción de expansión *(CALL expanded statement)*

DESCRIPCION

Esta instrucción hace una llamada a otra instrucción añadida, mediante un cartucho de ROM. Su abreviatura es "_", el carácter subrayado. La instrucción CALL depende del cartucho de ROM; por tanto, si la empleamos en algún programa, éste pierde la compatibilidad con el MSX estándar.

SINTAXIS

CALL <nombre>(<lista de argumentos>)
_<nombre>(<lista de argumentos>)

La lista de argumentos puede ser de cualquier tipo, dependiendo de lo que haga la instrucción de expansión.

EJEMPLO

CALL SPEECH (volumen %, voz %, "HELLO")

PUNTOS A RECORDAR

En otros ordenadores se emplea la instrucción CALL para ejecutar subrutinas en código máquina; en el MSX no se hace así. Para llamar a dichas subrutinas se emplea la instrucción USR.

La instrucción ampliada ha de tener quince caracteres como máximo.

La instrucción CALL no se empleará a menos que hayas conectado un cartucho a tu ordenador. Esta instrucción depende totalmente de la ROM.

PRECAUCIONES

No utilices esta instrucción en programas comerciales, ya que perdería la compatibilidad con otros ordenadores MSX.

Una llamada a una instrucción de expansión no existente te causará un error sintáctico *(Syntax error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 50: "Memoria y cartucho del MSX".

CDBL Conversión en número de DoBLe precisión

DESCRIPCION

La función CDBL convierte un número entero, de precisión simple, variables o expresiones, a números de doble precisión.

SINTAXIS

CDBL(<const-num>)
CDBL(<var-num>)
CDBL(<exp-num>)

EJEMPLOS

B# = CDBL(A!)
PRINT CDBL(NO%)

PUNTOS A RECORDAR

Los números de doble precisión tienen una longitud de 8 bytes, lo que hace que puedan tener catorce cifras decimales.

PRECAUCIONES

Si el argumento de la instrucción es una cadena de caracteres se producirá un error de tipo de datos *(Type mismatch error)*; también ocurrirá cuando se asigne una instrucción CDBL a una variable de tipo cadena de caracteres.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CINT
CSNG

Parte segunda, tema 31: "Conversión de tipos".

CHR$ (Cadena de caracteres)
(CHaRacter string)

DESCRIPCION

Dicha función nos devuelve el carácter correspondiente al código ASCII dado como argumento. Se trata de la función inversa a la función ASC.

Partiendo de la base de que algunos códigos ASCII se corresponden con caracteres de control (por ejemplo, borrar la pantalla), podemos activarlos mediante PRINT CHR$.

SINTAXIS

CHR$(<const-num>)
CHR$(<var-num>)
CHR$(<exp-num>)

EJEMPLO

PRINT CHR$(66) nos mostrará la letra "B" en pantalla.

```
10 C1=83 : REM El codigo ASCII de S es 83
20 C2=65 : REM El codigo ASCII de A es 65
30 C3=76 : REM El codigo ASCII de L es 76
40 PRINT CHR$(C1);CHR$(C2);CHR$(C3)
50 A$=CHR$(C1+32)+CHR$(C2+32)+CHR$(C3+32)
60 PRINT A$
```

Si ejecutas este programa te darás cuenta que, sumando 32 al código ASCII de las letras mayúsculas, transforma a éstas en minúsculas.

CINT (Conversión a entero)
(Convert to INTeger)

DESCRIPCION

Esta función transforma números de precisión simple y doble, variables y expresiones numéricas, a números enteros.

Su rango está comprendido entre −32768 y 32767; cualquier argumento fuera de rango te causará problemas.

Si el argumento de esta función es un número real, el resultado será el mismo número con su parte decimal truncada.

EJEMPLO

```
A%=CINT(B+C)
PRINT CINT(1234.56789) le mostrará 1234 en pantalla.
```

PUNTOS A RECORDAR

Cuando el argumento de esta función esté comprendido entre 0 y 31, CHR$ actuará de acuerdo con el carácter de control correspondiente en el código ASCII estándar. Por ejemplo, si haces PRINT CHR$(7) obtendrás un *beep* del altavoz. Consulta la tabla de códigos de caracteres de control. Te darás cuenta de que puedes introducir códigos de control dentro de una cadena de caracteres.

Sin embargo, si el argumento se encuentra entre 32 y 255, la función te devolverá el carácter correspondiente al código ASCII dado.

PRECAUCIONES

Si el argumento de la función es negativo o mayor que 255 se producirá un error de llamada ilegal a una función *(Illegal function call)*.

Recuerda que el resultado de la función CHR$ es un carácter y que no puede ser igualado a una variable de tipo numérico, ya que ocasionaría un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Véanse los apéndices para la tabla de códigos de caracteres ASCII.
Parte primera, tema 20: "Los códigos ASCII".

CINT Conversión a entero *(Convert to INTeger)*

DESCRIPCION

Esta función transforma números de precisión simple y doble, variables y expresiones numéricas, a números enteros.

Su rango está comprendido entre −32768 y 32767; cualquier argumento fuera de rango te causará problemas.

Si el argumento de esta función es un número real, el resultado será el mismo número con su parte decimal truncada.

EJEMPLO

A%=CINT(B!*C)

PRINT CINT(1234.56789) te mostrará 1234 en pantalla.

PUNTOS A RECORDAR

La longitud de los números enteros es de 2 bytes.

PRECAUCIONES

El argumento de la función ha de estar comprendido entre −32768 y 32767. Si el argumento se encontrase fuera de rango se produciría un error de desbordamiento *(Overflow error)*.

Si el argumento de la función es de tipo cadena de caracteres o se lo asignas a una cadena de dicho tipo, te causará un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CDBL
CSNG
FIX

Parte segunda, tema 31: "Conversión de tipos".

CIRCLE Traza un círculo *(draw a CIRCLE)*

DESCRIPCION

La instrucción CIRCLE dibuja un círculo o una elipse, o bien una parte de ellos, siempre que estemos en el modo gráfico. Puedes indicar el centro del círculo, el color e incluso qué parte del círculo quieres que te dibuje el ordenador.

SINTAXIS

CIRCLE <especificación-coordenadas>,<radio>,<color>,<ángulo de comienzo>,<ángulo de fin>,<variación del radio>.

Estos parámetros pueden ser <const-num>, <var-num> o <exp-num>, pero se han de encontrar dentro de su rango correspondiente.

Explicación de <especificación-coordenadas>:

Hay dos modos de especificar las coordenadas del centro, uno en modo relativo y otro en modo absoluto.

<especificación-coordenadas> puede ser cualquiera de los términos siguientes:

1) (<coordenada x>,<coordenada y>)

Estas coordenadas especifican el centro del círculo en la pantalla en modo absoluto.

2) STEP(<parámetro x>,<parámetro y>)

Las coordenadas <parámetro x>, <parámetro y>, se obtienen a partir del último punto dibujado por el ordenador. Si el punto resultante queda fuera de la pantalla, el ordenador pintará hasta los bordes.

<color>, <ángulo de comienzo>, <ángulo de fin> y <variación del radio> son opcionales.

Rangos:

<coordenada x>	Entre −32768 y 32767. Para situar el centro dentro de la pantalla se ha de emplear el rango 0-255.
<coordenada y>	Entre −32768 y 32767. Para situar el centro dentro de la pantalla se ha de emplear el rango 0-191.
<radio>	Entre 0 y 32767; pero el círculo no se ajustará a la pantalla si el <radio> es demasiado grande.

<color> Este 0 y 15. (Véanse los códigos de color en el tema "Color".)

<ángulo de comienzo> Entre 0 y 2*PI radianes.

<ángulo de fin> Entre 0 y 2*PI radianes.

Si <ángulo de comienzo> y <ángulo de fin> son negativos dentro de su rango, se dibujará una recta desde el centro hasta el extremo.

<variación del radio> entre 0 y 32767.

<variación del radio> = radio vertical/radio horizontal.

Si la variación del radio es mayor que 1, la elipse se abombará verticalmente; en cambio, si es menor que 1 se abombará horizontalmente.

Si no se especifica la variación del radio el ordenador la tomará como 1, dibujando así un círculo.

Caso de que la variación del radio sea muy grande obtendrás una línea vertical; en cambio, si es 0 obtendrás una línea horizontal.

EJEMPLO

```
10 SCREEN 2
20 CIRCLE (100,100),50,8
30 FOR I=1 TO 5
40 CIRCLE STEP(I*5,0),50,8
50 NEXT
60 GOTO 60: REM Mantiene este modo de pantalla
```

PUNTOS A RECORDAR

Esta instrucción sólo es válida en los modos gráficos. Asegúrate que estás en SCREEN 2 o en SCREEN 3.

Si no especificas el color, el ordenador tomará el color del texto en ese momento.

Los ángulos de comienzo y fin del círculo se especifican en radianes.

PRECAUCIONES

Si <especificación-coordenadas> o <radio> son valores fuera de rango, te producirá un error de desbordamiento *(Overflow error)*.

En cambio, si es el <color> el que está fuera de rango el error producido será el de llamada ilegal a una función *(Illegal function call)*; este mismo error se producirá si los ángulos de comienzo y de fin están fuera de su rango correspondiente.

COLOR
SCREEN

Parte primera, tema 25: "Círculos y elipses"

La instrucción CLEAR indica al ordenador que borre todas las variables que se encuentren en ese momento almacenadas en memoria. Borra todo el espacio de variables en memoria, anulando así todas las variables numéricas y de cadena de caracteres.

También puedes reservar memoria para variables de tipo cadena de caracteres y código máquina.

El espacio de memoria se reserva cambiando el valor de la dirección más alta de memoria disponible por el BASIC. La dirección más alta de memoria tiene el valor &HF380. La puedes reducir mediante CLEAR, <tope-mem>. El espacio creado al reducir la memoria BASIC lo puedes emplear para subrutinas en código máquina, así como para datos.

SINTAXIS

CLEAR <espacio caract>	Cambia el tamaño dedicado a cadenas de caracteres.
CLEAR ,<tope-mem>	Cambia la dirección más alta de memoria permitida al BASIC.
CLEAR <espacio caract>, <tope-mem>	Cambia el tamaño dedicado a cadenas de caracteres y la dirección más alta permitida al BASIC.

EJEMPLOS

CLEAR	
CLEAR 255	Amplía el área de memoria dedicada a cadenas de caracteres a un espacio de 255 bytes por cadena.
CLEAR ,&HF300	Crea espacio para programas en código máquina entre las direcciones &HF300 y &HF380.

PUNTOS A RECORDAR

El espacio inicial dedicado a cadenas de caracteres es de 200 bytes, con lo que se fija a dichas cadenas una longitud máxima de 200 caracteres.

Si deseas que esta longitud se amplíe debes ejecutar antes de nada la instrucción CLEAR 255.

La memoria comprendida entre las direcciones &HF3FF y &HF380 está considerada como área de trabajo del sistema por la ROM dedicada al BASIC.

CLEAR borrar área de memoria *(CLEAR memory area)*

DESCRIPCION

La instrucción CLEAR indica al ordenador que borre todas las variables que se encuentren en ese momento almacenadas en memoria. Borra todo el espacio de variables en memoria, anulando así todas las variables numéricas y de cadena de caracteres.

También puedes reservar memoria para variables de tipo cadena de caracteres y código máquina.

El espacio de memoria se reserva cambiando el valor de la dirección más alta de memoria disponible por el BASIC. La dirección más alta de memoria tiene el valor &HF380. La puedes reducir mediante CLEAR, <tope-mem>. El espacio creado al reducir la memoria BASIC lo puedes emplear para subrutinas en código máquina, así como para datos.

SINTAXIS

CLEAR <espacio caract>	Cambia el tamaño dedicado a cadenas de caracteres.
CLEAR ,<tope-mem>	Cambia la dirección más alta de memoria permitida al BASIC.
CLEAR <espacio caract>, <tope-mem>	Cambia el tamaño dedicado a cadenas de caracteres y la dirección más alta permitida al BASIC.

EJEMPLOS

CLEAR	
CLEAR 255	Amplía el área de memoria dedicada a cadenas de caracteres a un espacio de 225 bytes por cadena.
CLEAR ,&HF300	Crea espacio para programas en código máquina entre las direcciones &HF300 y &HF380.

PUNTOS A RECORDAR

El espacio inicial dedicado a cadenas de caracteres es de 200 bytes, con lo que se fija a dichas cadenas una longitud máxima de 200 caracteres.

Si deseas que esta longitud se amplíe debes ejecutar antes de nada la instrucción CLEAR 255.

La memoria comprendida entre las direcciones &HFFFF y &HF380 está considerada como área de trabajo del sistema por la ROM dedicada al BASIC.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

Parte segunda, tema 30: "Constantes y variables".

Parte segunda, tema 48: "Mapa de memoria".

CLOAD/CLOAD? Carga desde el cassette (*LOAD from cassette*)

DESCRIPCION

Para cargar un programa en BASIC desde el cassette usa la instrucción CLOAD. Cuando el ordenador encuentre un programa te indicará si lo está cargando en memoria o si lo está saltando. No necesitas conocer el nombre del primer programa BASIC en cinta. Si es éste el que deseas cargar, basta con omitir el nombre del programa al ejecutar la instrucción.

La buena o mala carga de un programa depende exclusivamente de la calidad de tu aparato de cassette.

CLOAD? comprueba el programa grabado en cinta con el que está cargado en memoria. El ordenador te dará un mensaje (O.K.) si los programas son iguales, o bien te dará un error de comprobación si ambos difieren.

SINTAXIS

CLOAD "<nombre>"
CLOAD?
CLOAD? "<nombre>"

EJEMPLOS

Supón que tenemos dos programas llamados "VIBORA" y "ESCALA" grabados en cinta. Si tecleas:

CLOAD<return>

```
CLOAD
Found : VIBORA
OK
```

Ahora teclea:

CLOAD "ESCALA"<return>

```
CLOAD "ESCALA"
Skip  : VIBORA
Found : ESCALA
OK
```


PUNTOS A RECORDAR

El nombre del programa no puede tener más de seis caracteres.

Cualquier programa que tengamos almacenado en memoria se borrará si cargamos un programa del cassette.

La función de la tecla F7 es CLOAD.

La línea de transmisión de cassette y memoria puede ser de 1.200 ó 2.400 baudios; el ordenador detectará la velocidad de transmisión y actuará de acuerdo con ella.

Si hubiese algún fichero abierto en el cassette la instrucción CLOAD lo cerraría automáticamente y continuaría ejecutando la carga en memoria.

PRECAUCIONES

Si intentas cargar en memoria un programa grabado defectuosamente se producirá un error del dispositivo de E/S *(Device I/O Error)*. Emplea siempre un cassette fiable.

No conseguirás cargar nada si no tienes el nombre correcto del fichero; por tanto, has de ser cuidadoso con los nombres de los ficheros.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CSAVE

Parte primera, tema 11: "Grabación de programas en cassette".

CLOSE Cerrar canal *(CLOSE channel)*

DESCRIPCION

Esta instrucción cierra un canal abierto anteriormente. Es la instrucción contraria a OPEN. El *buffer* correspondiente a dicho canal también será liberado.

SINTAXIS

CLOSE ... cierra todos los canales.
CLOSE # <num-fichero>,<num-fichero>

EJEMPLO

CLOSE #2

PRECAUCIONES

Si el argumento es un número de fichero inexistente se producirá un error de número de fichero equivocado *(Bad file Number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OPEN

Parte segunda, tema 47: "Manejo de ficheros".

CLS Borrar pantalla *(CLear the Screen)*

DESCRIPCION

Estando la pantalla en cualquier modo, CLS la borra. En el modo gráfico la pantalla se borra permaneciendo el color del fondo.

SINTAXIS

CLS

EJEMPLOS

Las siguientes sentencias borran la pantalla poniéndola de color rojo claro:

```
10 SCREEN 2
20 COLOR ,9
30 CLS
40 GOTO 40
```

LINEA 10 SELECCIONA EL MODO GRAFICO DE ALTA RESOLUCION.
LINEA 20 FIJA EL COLOR DEL FONDO EN ROJO CLARO.
LINEA 30 BORRA LA PANTALLA.
LINEA 40 HACE FIJO EL MODO DE PANTALLA.

>Resultado: Pantalla en rojo claro.

PUNTOS A RECORDAR

El cursor se situará en la parte superior derecha de la pantalla.

Cuando las funciones asociadas a las teclas se visualizan con KEY ON, en la pantalla —en modo texto— se seguirá viendo la lista de funciones en la parte inferior, después de haberla borrado. Para borrar también dicha lista de funciones has de ejecutar KEY OFF.

Control L: <CTRL><L> tiene el mismo efecto que CLS.
PRINT CHR$(12) realiza la misma función que CLS; por tanto, se puede incluir dentro de una cadena de caracteres.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

COLOR

Parte primera, tema 3: "Escritura de un programa".

COLOR color

DESCRIPCION

Esta instrucción fija el color del texto, el papel y el fondo.
Hay quince colores a tu disposición, aparte de un color transparente.

SINTAXIS

COLOR <color-texto>, <color-papel>, <color-fondo>.

Todos los números indicadores del color pueden ser <exp-num>, debiendo estar comprendidos entre 0 y 15. Estos números son opcionales; por tanto, se pueden omitir al dar la orden.

Códigos de color:

0	Transparente	8	Rojo (medio)
1	Negro	9	Rojo (pálido)
2	Verde (medio)	10	Amarillo (oscuro)
3	Verde (claro)	11	Amarillo (claro)
4	Azul (oscuro)	12	Verde (oscuro)
5	Azul (claro)	13	Magenta
6	Rojo (oscuro)	14	Gris
7	Cian	15	Blanco

EJEMPLOS

COLOR 11	Fija el color del texto en amarillo claro.
COLOR 15,,15	Fija el color del texto y el fondo en blanco.
COLOR 1,5,1	Fija el color del texto y el fondo en negro, y el del papel, en azul claro.

PUNTOS A RECORDAR

Al conectar el ordenador, éste fija en pantalla unos colores estándar (COLOR 15,4,7)*; es decir, texto blanco, papel azul oscuro y fondo cian.

La tecla F-1 tiene como función asociada la instrucción COLOR. En cambio la tecla F-6 tiene como función asociada el color estándar; es decir, COLOR 15,4,7*.

El color del papel no cambia, excepto si se ejecuta la orden CLS.

Si en las instrucciones PSET, LINE o CIRCLE no se especifica el color, éstas tomarán aquél prefijado en la última instrucción COLOR.

* En las versiones europeas y americanas es 15,4,4.

PRECAUCIONES

El argumento de la función ha de estar compendido entre 0 y 15; en caso contrario se producirá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DRAW
PSET
PRESET
LINE
CIRCLE
CLS

Parte primera, tema 22: "Color".

Parte segunda, tema 41: "Gráficos avanzados (II): el color en los gráficos de alta resolución de MODO 2".

CONT CONTinua *(CONTinue)*

DESCRIPCION

Este comando indica al ordenador que continúe ejecutando a partir del punto en que se quedó parado, bien mediante una instrucción STOP o bien tecleando <CTRL><STOP>. El ordenador memoriza la última línea ejecutada antes de haber hecho <CTRL><STOP>, y continúa a partir de la siguiente línea. Si la última instrucción ha sido un INPUT, entonces continuaría la ejecución a partir de ella.

SINTAXIS

CONT

EJEMPLO

```
10 PRINT "CUANDO ES TU CUMPLEANOS?"
20 INPUT AND,MES,DIA
30 PRINT "GRACIAS"
```

```
CUANDO ES TU CUMPLEANOS?
?1970
??13        pulsa <CTRL><STOP>
BREAK IN 20
OK
CONT
?1970
??12
??1
GRACIAS
OK
```

PUNTOS A RECORDAR

La instrucción CONT se ejecutará exclusivamente en los siguientes casos:

1) Después de que un programa haya detenido su ejecución mediante una instrucción STOP. El programa seguirá la ejecución a partir de la siguiente línea.
2) Después de que el programa se hubiese detenido como consecuencia de una instrucción END. El programa continuará a partir de la siguiente línea.
3) Después de haber detenido la ejecución del programa mediante <CTRL><STOP>. El programa seguirá ejecutándose a partir de la siguiente línea.

PRECAUCIONES

La instrucción CONT no tendrá validez alguna en los siguientes casos:

1) Después de la edición del programa.
2) Después de parar la impresora.
3) Después de interrumpir la ejecución de comandos de entrada/salida del cassette, como, por ejemplo, INPUT#.

El mensaje de error que se obtendrá en estos casos será el de la imposibilidad de proseguir una ejecución *(Cant' Continue)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

COS COSeno

DESCRIPCION

La función COS nos devuelve el valor del coseno del ángulo dado como argumento.

El valor obtenido mediante esta función es de doble precisión.

El ángulo del cual queremos calcular su coseno ha de ir expresado en radianes. El MSX no reconoce los ángulos en grados.

SINTAXIS

COS(<const-num>)
COS(<var-num>)
COS(<exp-num>)

EJEMPLOS

```
PRINT COS(1.4445432)
.12591798465524
```

PUNTOS A RECORDAR

El valor que nos devuelve la función COS es de doble precisión. El argumento de la función ha de estar en radianes.

PRECAUCIONES

Esta es una función trigonométrica; por tanto, su argumento no puede ser una cadena de caracteres ni tampoco puede estar asignado a otra cadena del mismo tipo. Si, por ejemplo, ejecutas COS(A$), te producirá un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SIN
TAN
ATN

Parte primera, tema 19: "Funciones matemáticas".

CSAVE Grabar en cassette *(Cassette SAVE)*

DESCRIPCION

Esta orden graba de la memoria al cassette un programa en BASIC. Al programa se le debe dar un nombre que tenga —como máximo— seis caracteres. Para abreviar tiempo los programas se graban en *tokens* y no en forma de ficheros ASCII. Se puede especificar la capacidad de transmisión del canal; ésta podrá ser de 1.200 ó 2.400 baudios. Si dicha capacidad se omite el ordenador tomará por defecto 1.200 baudios.

SINTAXIS

CSAVE "<nombre>"
CSAVE "<nombre>", cap-trans

donde <cap-trans> puede ser:

1 = 1.200 baudios
2 = 2.400 baudios.

EJEMPLOS

```
CSAVE "AVENTU"
CSAVE "juegos",2
```

PUNTOS A RECORDAR

El nombre del programa ha de tener como máximo seis caracteres.

La capacidad de transmisión también se puede fijar con la orden SCREEN.

Los programas grabados en cinta cassette con el comando CSAVE no pueden ser montados con otros, ya que han sido grabados en forma de *tokens*. Para poder montar dos programas, al menos uno de ellos ha de estar almacenado en cassette con el comando SAVE, que graba los ficheros en código ASCII.

PRECAUCIONES

La perfecta grabación de un programa depende de la calidad de la cinta y el buen estado del aparato cassette.

Si el programa se encuentra mal grabado en cinta e intentamos cargarlo en memoria con CLOAD nos producirá un error de dispositivo de *(Device I/O error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CLOAD
SAVE

Parte primera, tema 11: "Grabación de programas en cassette".

CSNG Conversión a precisión simple *(Convert to SiNGle precision number)*

DESCRIPCION

Esta función convierte el número dado como argumento a precisión simple.

SINTAXIS

CSNG(<const-num>)
CSNG(<var-num>)
CSNG(<exp-num>)

EJEMPLOS

```
PRINT COS(0.7656)
   .72096670541357
PRINT CSNG(COS(0.7656)
   .720967
```

PUNTOS A RECORDAR

La función CSNG redondea el argumento a partir del sexto decimal.

PRECAUCIONES

Si el argumento de la función fuese de tipo cadena de caracteres producirá un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CDBL
CINT

Parte segunda, tema 31: "Conversión de tipos".

CSRLIN LINea en que se encuentra el CuRSor

DESCRIPCION

Esta variable del sistema nos permite conocer la línea en la que se encuentra el cursor de texto.

SINTAXIS

CSRLIN

EJEMPLO

```
PRINT CSRLIN
```

PUNTOS A RECORDAR

CSRLIN tiene el valor 0 para la línea superior.

PRECAUCIONES

No se puede asignar ningún valor a esta variable perteneciente al sistema. Si lo intentas te producirá un error de sintaxis *(Syntax error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

POS(0)

Parte segunda, tema 40: "Gráficos avanzados (I): características de cada modo de pantalla".

DATA

DESCRIPCION

En las instrucciones DATA puedes almacenar para tu programa en BASIC todos los datos que desees; el acceso a éstos se realiza mediante la instrucción READ. En estas instrucciones puedes incluir todos los números y cadenas de caracteres que quieras, separados por comas (,), mientras no rebases la capacidad de una línea del programa. La posición de estas instrucciones DATA dentro de un programa no es importante, ya que el ordenador las ignora cuando encuentra una; por tanto, las puedes situar en cualquier zona de tu programa.

El ordenador accede a los datos de estas instrucciones mediante la instrucción READ. La lectura de los datos se realiza secuencialmente; el ordenador recuerda el último dato leído y en la siguiente orden de lectura (READ) accede al dato posterior al último leído.

Los datos de una instrucción DATA pueden ser numéricos o cadenas de caracteres. La longitud de la instrucción no puede superar los 255 caracteres, debido a la ocupación máxima permitida de una línea. Los números pueden ser enteros —de precisión simple o doble—. Las cadenas de caracteres no deben ir entre comillas —" "—, excepto si llevan comas, dos puntos, punto y coma, etc. Dentro de las sentencias DATA no se pueden incluir ni variables ni expresiones.

La variable correspondiente de la instrucción READ ha de ser del mismo tipo que el dato de la instrucción DATA; por ejemplo, una variable de tipo cadena de caracteres con un dato del mismo tipo.

SINTAXIS

DATA <lista de datos>

EJEMPLO

```
10 PRINT "LISTIN TELEFONICO"
20 FOR I=1 TO 4
30 READ NOM$,PREFI,TEL
40 PRINT NOM$,PREFI;"-";TEL
50 NEXT I
60 DATA JULIO,091,4397867,ANTONIO,093,832475
70 DATA PEPA,087,2223344,SERAFINA,918,3332211
```



```
LISTIN TELEFONICO
JULIO          91 - 4397867
ANTONIO        93 - 832475
PEPA           87 - 2223344
SERAFINA       918 - 3332211
```

PUNTOS A RECORDAR

Las cadenas de caracteres han de ir entre comillas en los siguientes casos:

1) Si la cadena tiene comas.
2) Si la cadena tiene dos puntos.
3) Si la cadena tiene espacios en blanco, antes o después de los caracteres.

Para acceder a una instrucción DATA en particular has de emplear la instrucción RESTORE; de esta manera podrás acceder (READ) al primer dato de la instrucción DATA especificada con RESTORE.

PRECAUCIONES

Si los tipos de la variable de la instrucción READ y el dato de la instrucción DATA no son iguales se producirá un error en el tipo de datos *(Type mismatch error)*. Por tanto, las variables numéricas han de corresponderse con números, y las variables de tipo cadena de caracteres, con cadenas de caracteres.

Si después de haber leído el último dato intentas leer alguno más y no has ejecutado la orden RESTORE, se producirá un error de intento de lectura sin haber datos *(Out of data)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

READ
RESTORE

Parte primera, tema 12: "Lectura de datos de matrices".

DEFDBL DEFine DoBLe precisión

DESCRIPCION

Esta instrucción declara los caracteres con los que comienzan los nombres de las variables de doble precisión.

SINTAXIS

DEFDBL<rango(s) de letras>

EJEMPLO

DEFDBL A, B, C Declara todas las variables que empiezan por A, B o C como numéricas de doble precisión.

PUNTOS A RECORDAR

Todas las declaraciones de tipo mediante los sufijos #, $, % y ! tienen preferencia sobre la declaración DEFDBL.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DEFINT
DEFSNG
DEFSTR

Parte segunda, tema 30: "Constantes y variables".

DEF FN DEFinición de una FuNción

DESCRIPCION

Mediante esta instrucción puedes definir tus propias funciones a las cuales puedes llamar desde cualquier parte del programa. Esta instrucción se ha creado para reservar una zona de memoria a expresiones que se emplean frecuentemente a lo largo de un programa.

Puedes dar a tus funciones el nombre que desees, mientras que éste no coincida con ninguna de las palabras reservadas del ordenador.

La instrucción DEF FN ha de ir declarando a la función antes de que se haga uso de ella en cualquier parte del programa.

Puede hacer corresponder la función declarada a todos los parámetros que desees.

Cualquier variable que está incluida en la función es local, y desde fuera de la propia función no se la puede cambiar el valor que tenga asignado.

Para efectuar una llamada a la función declarada has de situar delante del nombre la instrucción FN.

La declaración de una función mediante DEF FN puede ser todo lo extensa que quieras mientras que quepa en una línea.

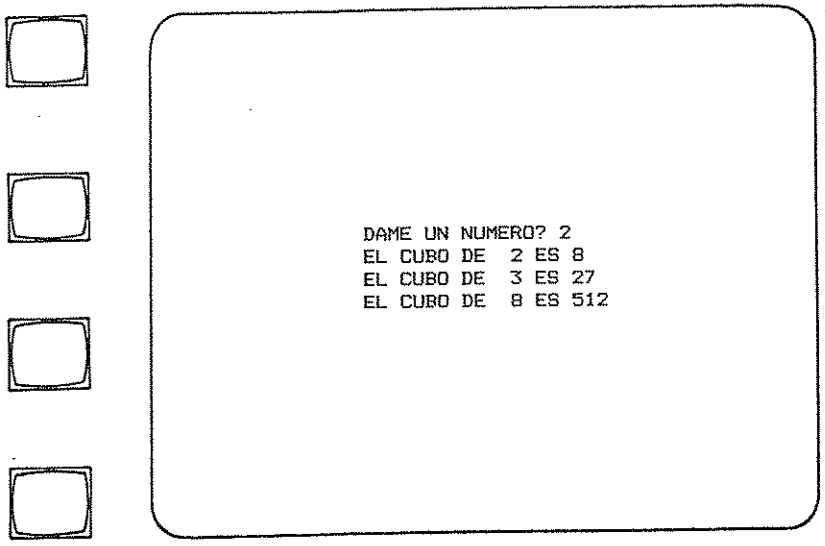
SINTAXIS

DEF FN<nombre> = <expresión>
DEF FN<nombre>(<parámetros>)= <expresión>

Si la función operase con cadenas de caracteres debe llevar el carácter "$" al final del nombre. Se pueden añadir, si esto fuese necesario, los sufijos de declaración de tipo al final del nombre.

EJEMPLO

```
10 DEF FNCUBO(X)=X^3
20 INPUT "DAME UN NUMERO";A%
30 PRINT "EL CUBO DE ";A%;"ES";FNCUBO(A%)
40 B%=FNCUBO(A%+1)
50 PRINT "EL CUBO DE ";A%+1;"ES";B%
60 C%=FNCUBO(FNCUBO(A%))
70 PRINT "EL CUBO DE ";FNCUBO(A%);"ES";C%
```



PUNTOS A RECORDAR

La lista de parámetros que pasan a la función pueden ser numéricos, cadenas de caracteres o expresiones, siempre y cuando vayan separados por comas.

En la declaración de la función se pueden incluir variables que se usen fuera de la función.

Se pueden encadenar funciones, como en el ejemplo anterior, pasando las funciones como parámetros de otra función.

PRECAUCIONES

La mayoría de los errores que se producen ocurren en el momento de llamada. Aquí tienes algunos de ellos:

1) Función de usuario no definida *(Undefined user function)*, que se producirá si haces una llamada a la función antes de la declaración DEF FN.
2) Si los tipos de los parámetros no se corresponden con los de la llamada a la función, o el resultado de la función se encuentra asignado a una variable de distinto tipo, tendrás un error en el tipo de datos *(Type mismatch error)*; por ejemplo, A$ = FN CUBO(2).
3) También se producirá error si pasas parámetros distintos en una llamada que aquéllos de la declaración.
4) Error en el tipo de datos *(Type mismatch error)* puede estar ocasionado por no ser compatibles los tipos declarados de los parámetros dentro de la declaración de función.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 16: "Tus propias funciones".

DEFINT Declaración de números enteros *(DEFine INTeger)*

DESCRIPCION

La instrucción DEFINT declara que todas las variables que comienzan con el rango de letras declarado son numéricas enteras.

SINTAXIS

DEFINT<rango de letras>

EJEMPLO

DEFINT I, J, K	Declara que todas las variables que comiencen por I, J o K son variables enteras.

PUNTOS A RECORDAR

La declaración de tipo de variable mediante los sufijos #, $, % y ! tienen preferencia sobre la declaración DEFINT.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DEFDBL
DEFSNG
DEFSTR

Parte segunda, tema 30: "Constantes y variables".

DEFSNG Declaración de números de precisión simple *(DEFine SiNGle precision)*

DESCRIPCION

Esta instrucción declara que cualquier variable que comience con alguno de los caracteres especificados en su argumento es numérica de precisión simple.

SINTAXIS

DEFSNG <rango(s) de letras>

EJEMPLO

DEFSNG X, Y, Z — Declara que las variables que comiencen por X, Y o Z son variables numéricas de precisión simple.

PUNTOS A RECORDAR

La declaración de tipos de variables empleando los sufijos #, $, % y ! tienen preferencia sobre la declaración DEFSNG.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DEFDBL
DEFINT
DEFSTR

Parte segunda, tema 30: "Constantes y variables".

DEFSTR Declaración de cadenas de caracteres *(DEFine STRing)*

DESCRIPCION

Con esta instrucción se declara que todas las variables que comiencen con alguno de los caracteres indicados son del tipo cadena de caracteres.

SINTAXIS

DEFSTR <rango(s) de caracteres>

EJEMPLO

DEFSTR E, R, T

PUNTOS A RECORDAR

La declaración de tipos de variables mediantes sufijos #, $, % y ! tiene preferencia sobre la declaración DEFSTR.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DEFDBL
DEFINT
DEFSNG

Parte segunda, tema 30: "Constantes y variables".

DEF USR DEFinición de la dirección de comienzo de una subrutina del USuaRio en código máquina

DESCRIPCION

Para llamar a una subrutina en código máquina desde el BASIC se emplea la instrucción USR; dicha subrutina comienza en la posición especificada por DEF USR.

Se pueden definir diez funciones de tipo USR como máximo. Cada una de ellas ha de ir seguida de un número; en este ejemplo el número es 5:

```
DEFSTR=&HFF80
```

Cuando quieras llamar a la subrutina declarada en DEFUSR tienes que emplear la instrucción USR; por ejemplo:

```
PRINT USR5 ("parámetro")
```

donde el parámetro entre paréntesis tiene un valor que va a pasar desde el BASIC a la subrutina en código máquina.

SINTAXIS

```
DEFUSR=<dirección-comienzo>, en caso de que el número sea 0.
DEFUSR<número>=<dirección-comienzo>
```

donde <dirección-comienzo> puede ser <const-num>, <var-num> o <exp-num>, pero siempre un entero.

<número> ha de estar comprendido entre 0 y 9 inclusive.

EJEMPLOS

```
DEFUSR5=&HF300
DEFUSR=&HF300 + 255
```

Para llamar a la subrutina en código máquina 5 debes hacer:

```
CHISPA = USR5(9):   (9) es el parámetro que se pasa a la subrutina en
                    código máquina.
```


PUNTOS A RECORDAR

Puedes tener, en un determinado momento, hasta diez subrutinas en código máquina; pero como se pueden redefinir, podrás tener todas las que desees.

DEFUSR es equivalente a DEFUSR0.

Has de conocer el código máquina del microprocesador Z-80 para poder llegar a usar las instrucciones USR y DEFUSR.

PRECAUCIONES

La dirección de comienzo de la subrutina, además de ser un número entero, ha de pertenecer a la RAM; si no fuera así, producirías un error.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 49: "Funciones USR y el código máquina".

DELETE

DESCRIPCION

Este es un comando del editor que borra la parte del programa que le indiques.

SINTAXIS

DELETE <línea>
DELETE <línea>-<línea>
DELETE-<línea>

EJEMPLOS

DELETE 20	Borra la línea 20.
DELETE 20-100	Borra desde la línea 20 a la 100, ambas inclusive.
DELETE -100	Borra desde la primera línea hasta la 100, inclusive.

PUNTOS A RECORDAR

Si sólo quieres borrar una línea, te será más fácil teclear el número de la línea y pulsar <return>.

El comando DELETE no se puede incluir dentro de los programas.

PRECAUCIONES

Si intentas borrar una línea que no existe se producirá un error de llamada a una función ilegal *(Illegal function call)*.

Asegúrate de que sólo borras las líneas que no necesitas, ya que una línea borrada es irrecuperable.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

Parte segunda, tema 29: "Edición avanzada de programas".

DIM Matriz DIMensional

DESCRIPCION

La instrucción DIM se emplea para declarar matrices. Una matriz es un grupo de variables con el mismo nombre, pero con diferente subíndice.

SINTAXIS

DIM <variable>(<const-num>)
DIM <variable>(<const-num>,<const-num>...)

<variable> es el nombre de la matriz declarada; ésta puede ser de tipo entero, de simple o doble precisión, o de tipo cadena de caracteres. <const-num> es el subíndice de mayor grado que puede tener la matriz, ha de ser un número entero positivo. Para declarar una matriz con más de un subíndice basta con añadir más <const-num> separadas por comas entre el paréntesis.

EJEMPLOS

DIM A(5,5,5) es una matriz de tres dimensiones.
DIM B$(100),C$(100,2),D%(100),E!(100),F#(100)

En una sola instrucción DIM puedes declarar todo tipo de matrices que desees.

PUNTOS A RECORDAR

Si el ordenador encuentra a lo largo del programa una matriz no declarada en una instrucción DIM, automáticamente se asume que la matriz tiene diez elementos. Si una matriz no declarada supera el subíndice máximo permitido (subíndice 10), producirá un error de rango en el subíndice. Así pues, no es preciso que definas matrices de menos de diez elementos.

La dimensión máxima de una matriz es de 255.

El valor mínimo del subíndice de una matriz es 0; por tanto, una matriz comienza con el elemento 0.

DIM A(5) es una matriz de seis elementos: A(0), A(1), A(2), A(3), A(4) y A(5).

Cuando declaramos una matriz todos sus elementos toman inicialmente el valor 0.

PRECAUCIONES

Si no te ajustas a la declaración de los índices de la instrucción DIM de una matriz producirás un error de índice fuera de rango *(Subscript out of range error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 12: "Lectura de datos de matrices".
Parte primera, tema 13: "Estructura y tratamiento de datos".
Parte segunda, tema 30: "Constantes y variables".

DRAW

DESCRIPCION

Esta instrucción te permite dibujar de acuerdo con el macrolenguaje gráfico (GML). Este macrolenguaje se escribe basándose en cadenas de caracteres; la instrucción DRAW indica al ordenador que dibuje según este sublenguaje.

Recuerda que existe un cursor gráfico que se puede desplazar por toda la pantalla. Mediante la programación en GML controlas los movimientos y acciones del cursor.

SINTAXIS

```
DRAW "<cad-car>"
DRAW  <cad-var>
DRAW  <cad-exp>
```

Comandos del macrolenguaje gráfico:

U<n>	: mueve el cursor gráfico hacia arriba.
D<n>	: mueve el cursor gráfico hacia abajo.
L <n>	: mueve el cursor gráfico hacia la izquierda.
R <n>	: mueve el cursor gráfico hacia la derecha.

<n> es, en cada caso, la distancia a mover; depende del factor de escala S (que se verá más adelante).

Cómo dibujar diagonales:

E<n>	: mueve el cursor diagonalmente hacia arriba a la derecha.
F <n>	: mueve el cursor diagonalmente hacia abajo a la derecha.
G<n>	: mueve el cursor diagónalmente hacia abajo a la izquierda.
H<n>	: mueve el cursor diagonalmente hacia arriba a la izquierda.

<n> es, en estos casos, algo diferente que cuando se dibuja horizontal o verticalmente. Por ejemplo, E<n> dibuja una recta entre el punto actual (X, Y) y el punto (X + <n>, Y + <n>).

Para dibujar una recta desde el punto actual hasta otro punto en concreto emplea el comando M.

M<X>,<Y>	: mueve el cursor desde el punto en que se encuentra hasta el punto X,Y especificado.

Si prefieres que las coordenadas hacia las que se mueve el cursor estén relacionadas con las del punto actual, basta con añadir un + o un − a la coordenada <X> e <Y>.

```
M+<X>,<Y>      : M−<X>,<Y>
M+<X>,−<Y>     : M−<X>,−<Y>
```

Cómo mover el cursor sin dibujar:

Hay ocasiones en que te interesa mover el cursor, pero no quieres que dibuje ese movimiento. Añadiendo el prefijo B, que viene del término inglés *blank* (vacío, blanco), consigues que el cursor se mueva sin dejar marca del camino seguido.

Comandos anteriores con el prefijo B:

B	: mueve el cursor sin dibujar.

Puedes emplear las siguientes combinaciones:

BU<n>	: mueve el cursor hacia arriba sin dibujar.
BD<n>	: mueve el cursor hacia abajo sin dibujar.
BL <n>	: mueve el cursor hacia la izquierda sin dibujar.
BR <n>	: mueve el cursor hacia la derecha sin dibujar.
BE <n>	: mueve el cursor hacia arriba a la derecha sin dibujar.
BF <n>	: mueve el cursor hacia abajo a la derecha sin dibujar.
BG<n>	: mueve el cursor hacia abajo a la izquierda sin dibujar.
BH<n>	: mueve el cursor hacia arriba a la izquierda sin dibujar.
BM<X>,<Y>	: mueve respecto al punto anterior con incremento X e Y.
BM+<X>,<Y>	
BM−<X>,<Y>	
BM+<X>,−<Y>	
BM−<X>,−<Y>	

Cómo volver a la posición anterior después de dibujar una línea:

Si después de dibujar una línea deseas volver al punto desde el cual la empezaste a dibujar, basta con que añadas el prefijo N a la instrucción que dibuja esa línea; esto hará que vuelvas a dicho punto.

Prefijo N	: Dibuja una línea y vuelve el cursor a la posición donde comenzó a dibujar.

Emplea las siguientes combinaciones para dibujar y volver al punto inicial:

NU <n>	: dibuja verticalmente hacia arriba y vuelve.
ND <n>	: dibuja verticalmente hacia abajo y vuelve.
NL <n>	: dibuja horizontalmente hacia la izquierda y vuelve.
NR <n>	: dibuja horizontalmente hacia la derecha y vuelve.
NE <n>	: dibuja en diagonal hacia arriba a la derecha y vuelve.
NF <n>	: dibuja en diagonal hacia abajo a la derecha y vuelve.
NG <n>	: dibuja en diagonal hacia abajo a la izquierda y vuelve.
NH <n>	: dibuja en diagonal hacia arriba a la izquierda y vuelve.
NM<X>,<Y>	: dibuja una línea hasta el punto (<X>,<Y>) y vuelve.
NM+<X>,<Y>	: dibuja una línea hasta el punto de coordenadas relativas de incrementos X e Y, y luego retorna al punto inicial.
NM-<X>,<Y>	
NM+<X>,-<Y>	
NM-<X>,-<Y>	

Rotación de los ejes:

Con el comando angular A puedes girar los ejes respecto a los cuales se refieren los dibujos realizados con los comandos U, D, L, R, E, F, G y H.

Comando angular A:

A<n>	: donde <n> puede ser 0, 1, 2 ó 3.
A0	: no giran los ejes.
A1	: giran 90° en sentido contrario a las agujas del reloj,
A2	: giran 180° en sentido contrario a las agujas del reloj.
A3	: giran 270° en sentido contrario a las agujas del reloj.

Cambios de color en el GML:

Con el comando C se cambia el color del cursor gráfico. Puedes cambiar el color tantas veces como desees con una simple instrucción DRAW.

Comando de color C:

C<n>	: controla el color mientras se dibuja.
<n>	: tiene que ser un número entero entre 0 y 15.

C0	Transparente	C8	Rojo
C1	Negro	C9	Rojo (claro)
C2	Verde	C10	Amarillo (oscuro)
C3	Verde (claro)	C11	Amarillo (claro)
C4	Azul (oscuro)	C12	Verde (oscuro)
C5	Azul (claro)	C13	Magenta
C6	Rojo (oscuro)	C14	Gris
C7	Cian	C15	Blanco

Cambio de la escala de dibujo:

Comando de escala S:

S<n>	: donde <n> es un entero entre 0 y 255. factor-escala = <n>,4

Por tanto, S1 dibuja 1/4 de la longitud especificada con los comandos U, D, L, R, etc.

S4 y S0 tienen el mismo efecto de no reducir a ninguna escala.

S8 fija la escala al doble del tamaño normal (S4).

Cómo emplear subcadenas:

X<cad-var>;

Esto significa que se ha de ejecutar el contenido de la cadena de caracteres. En este macrolenguaje puedes tener una cadena de caracteres que contenga los comandos de dibujo y luego pedir que se ejecute el contenido de dicha cadena. El punto y coma es siempre imprescindible.

```
DRAW"X<var-cad>;"
```

Cómo emplear variables numéricas en el GML:

En todos los comandos del lenguaje, <n>, <X> e <Y> pueden ser variables; en tal caso han de ir precedidos del símbolo de igualdad "=" y, a continuación, de un punto y coma. Es decir:

=<var-num>;	=<n>;
	=<X>;
	=<Y>;

Por ejemplo: sea la variable numérica G%; podemos hacer:

```
DRAW"U=G%;"
```

EJEMPLOS

```
10 SCREEN 2
20 A$="R50U50L50D50"
30 DRAW "BM100,150"
40 DRAW "S0XA$;"
50 IF INKEY$="" THEN 50
60 FOR I=1 TO 10
70 DRAW "S=I;XA$;"
80 NEXT I
90 GOTO 90
```

LINEA 10 SELECCIONA LA PANTALLA DE ALTA RESOLUCION.
LINEA 20 A$ CONTIENE EL MACROLENGUAJE GRAFICO.
LINEA 30 SITUA EL CURSOR GRAFICO EN EL PUNTO (100,150).
LINEA 40 DIBUJA UN CUADRADO EN TAMAÑO NORMAL.
LINEA 50 ESPERA QUE SE PULSE UNA TECLA PARA CONTINUAR.
LINEA 70 DIBUJA UN CUADRADO EN ESCALA 1/4.
LINEA 90 MANTIENE LOS DIBUJOS EN PANTALLA.

Nota: A$ hace "50 puntos a la derecha, 50 puntos arriba, 50 puntos a la izquierda, 50 puntos abajo».

PUNTOS A RECORDAR

El comando X resulta útil desde el momento en que puede dibujar parte de un modelo desde cualquier punto del programa. La idea es cómo tener una subrutina que dibuja lo que tú le pases como parámetros; o sea, los comandos del dibujo.

Con el comando X podrás realizar con una única instrucción DRAW dibujos con más de 255 caracteres de comandos, cuya ejecución sería imposible si los escribieras en forma explícita.

Comando S: si amplías el valor 4 de este comando el ordenador dibujará con el nuevo valor del factor de escala hasta que lo vuelvas a cambiar. Si deseas que vuelva el factor de escala normal, no tienes más que poner S4 en una instrucción DRAW.

BM sitúa el cursor gráfico en el punto especificado sin dibujar nada. Es el mejor modo de situar el cursor gráfico, aunque existen otras formas de hacerlo. Si no hay un comando BM, el ordenador comenzará a dibujar desde el último punto en que se situó.

Puedes situar el cursor en el punto de origen mediante la instrucción PSET o empleando el comando M.

Después de haberse ejecutado un comando A, angular, todo lo que se dibuje a continuación estará referido a los ejes rotados, excepto si retornas los ejes a la posición inicial con otro comando A.

El color del texto seguirá siendo el inicial después de haberse ejecutado una instrucción DRAW con un comando de color C; éste sólo tiene validez dentro de la instrucción DRAW.

PRECAUCIONES

Sólo puedes emplear la instrucción DRAW en modo gráfico.

Si hay algún error al escribir los comandos en GML producirás un error de llamada a una función ilegal *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

COLOR
SCREEN

Parte primera, tema 26: "El macrolenguaje gráfico".

ELSE

DESCRIPCION

ELSE forma parte de la instrucción estructurada IF/THEN/ELSE. Indica al ordenador que cuando no se cumpla la <condición> del IF salte las sentencias que siguen a THEN y ejecute las que siguen a ELSE.

ELSE GOTO<línea> puede simplificarse como ELSE<línea>. Puedes programar múltiples estructuras IF/THEN/ELSE, e incluso anidarlas. La única limitación que tienen es la de no sobrepasar los caracteres permitidos en una línea.

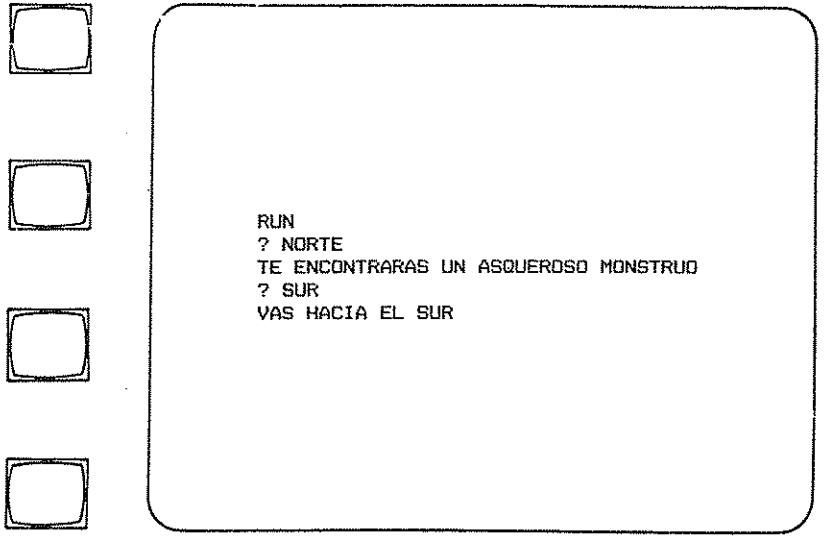
SINTAXIS

IF <condición> THEN <sentencias> ELSE <sentencias>
IF <condición> THEN <sentencias> ELSE <línea>

EJEMPLO

Aquí tienes un ejemplo de la sentencia IF/THEN/ELSE con estructura simple.

```
10 INPUT D$
20 IF D$="NORTE" THEN PRINT "TE ENCONTRARAS UN ASQ
UEROSO MONSTRUO" ELSE PRINT "VAS HACIA EL ";D$
30 GOTO 10
```



```
RUN
? NORTE
TE ENCONTRARAS UN ASQUEROSO MONSTRUO
? SUR
VAS HACIA EL SUR
```


PUNTOS A RECORDAR

Si en una sentencia IF/THEN/ELSE anidada encuentras menos ELSE que THEN, recuerda que cada ELSE va ligado al último THEN que le anteceda.

Por ejemplo:

```
IF THEN (IF THEN (IF THEN ELSE) ELSE)
```

Suele ocurrir que las sentencias IF/THEN/ELSE no quepan en una línea del programa, dado que a THEN y a ELSE les pueden seguir más de una instrucción; una solución a este problema es el uso de subrutinas mediante la orden GOSUB.

Si no utilizas con cuidado la sentencia IF/THEN/ELSE puedes provocar que tu programa se enrede.

PRECAUCIONES

Los problemas con estas instrucciones suelen estar en la <condición>.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

AND
OR
NOT
IF
THEN

Parte primera, tema 6: "Las condiciones".

Parte segunda, tema 35: "Algebra de Boole (II): la instrucción IF/THEN/ELSE".

END

DESCRIPCION

La instrucción END indica al ordenador que ha llegado al final del programa, devolviendo el control al modo directo.

Puedes incluir tantos END como desees, según te sea necesario, a lo largo del programa.

Una medida de precaución es la de poner una instrucción END antes de las subrutinas, evitando así que el ordenador las llegue a ejecutar secuencialmente, provocando resultados poco fiables.

Si el programa termina en la última línea el ordenador asume que allí existe una instrucción END; por tanto, no es necesario poner una instrucción END al final del programa.

Al ejecutarse una instrucción END se cierran automáticamente todos los ficheros que hubieran podido ser abiertos a lo largo del programa. Esta es una diferencia importante con STOP, que no cierra ningún fichero al detener la ejecución del programa.

SINTAXIS

END

EJEMPLO

```
9999 END
```

PUNTOS A RECORDAR

Otra diferencia entre STOP y END es que la primera manda un mensaje de "parada", mientras que la segunda no lo hace. La instrucción STOP —a diferencia de END— no cierra ningún fichero que pudiera estar abierto.

PRECAUCIONES

No pongas la instrucción END en ningún lugar del programa que no le corresponda, ya que éste finalizaría prematuramente.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

STOP

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programás".

EOF Fin de fichero *(End Of File)*

DESCRIPCION

Esta función te indica si has llegado al final de un fichero. Si has llegado al final de un fichero su valor −1 (verdadero), en caso contrario su valor será 0 (falso).

Esta función es muy útil para el manejo de ficheros, y se suele emplear junto con instrucciones como OPEN, etc.

SINTAXIS

EOF (<número-fichero>)

EJEMPLO

```
IF EOF(4) = -1 THEN PRINT "FIN DE FICHERO": END
```

PRECAUCIONES

Si tratas de leer de un fichero ya finalizado producirás un error de final de fichero ya leído; esto te ocurrirá posiblemente si no empleas la función EOF.

Si preguntas por el final de un fichero no abierto producirás un error de fichero no abierto *(File not open)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OPEN
CLOSE
INPUT

Parte segunda, tema 47: "Manejo de ficheros".

EQV EQuiValente

DESCRIPCION

La operación EQV es una de las funciones estándar del álgebra de Boole. Su función se muestra en la siguiente tabla de verdad:

EQV	X	Y	X EQV Y
	0	0	1
	0	1	0
	1	0	0
	1	1	1

Por ejemplo: 51 EQV 74

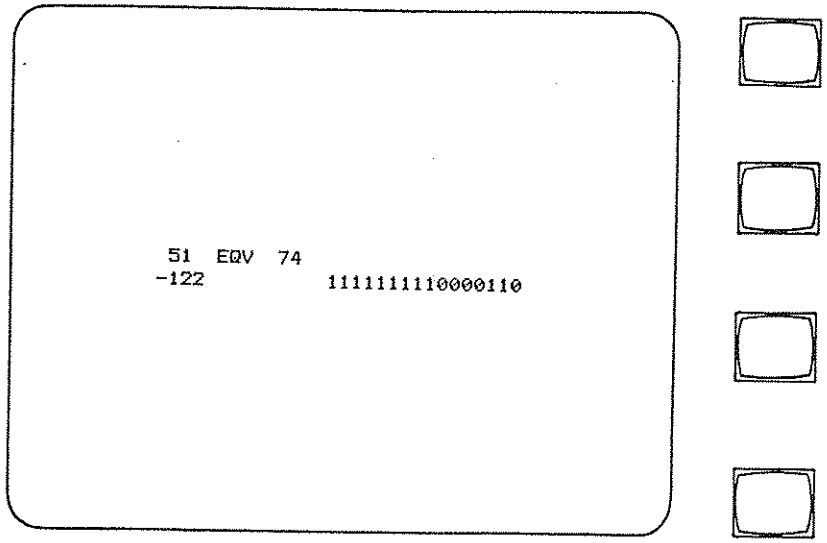
	51	0000000000110011
EQV	74	0000000001001010
	−122	1111111110000110

SINTAXIS

<número> EQV <número>
<número> tiene que ser un número entero comprendido entre −32768 y 32767. La parte decimal del <número> se redondea antes de ser operado.

EJEMPLO

```
10 A=51
20 B=74
30 PRINT A;" EQV ";B
40 PRINT A EQV B,BIN$(A EQV B)
```

PUNTOS A RECORDAR

Las siguientes igualdades se verifican:

X EQV X = −1
−X EQV Y = 1

PRECAUCIONES

Cuando uno de los operandos se sale de rango (−32768, 32767) se produce un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

AND
OR
XOR
IMP
NOT

Parte segunda, tema 34: "Algebra de Boole (I): operadores lógicos".

ERASE Borrar una matriz

DESCRIPCION

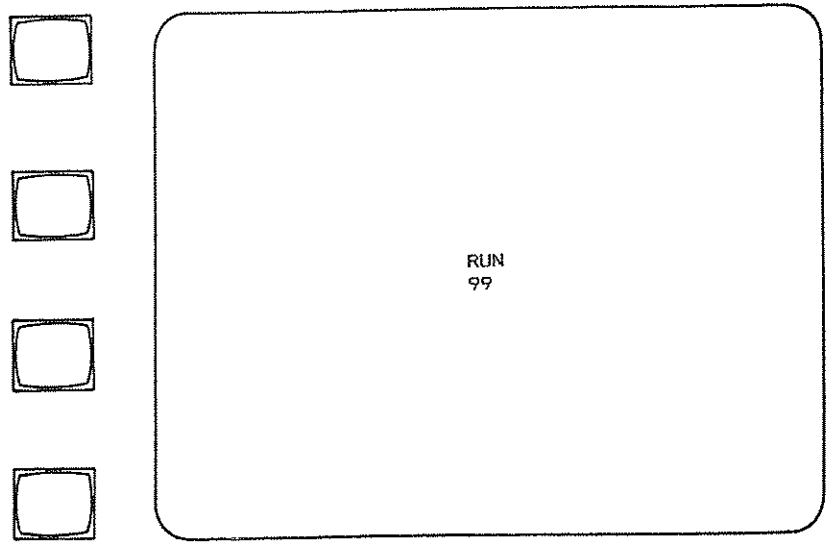
Esta instrucción te permite borrar una matriz creada mediante DIM; por tanto, puedes redimensionarla.

SINTAXIS

ERASE <nombre de la matriz>
ERASE <nombre de la matriz>,<nombre de la matriz>,...

EJEMPLO

```
10  A%=99
20  DIM A%(150)
30  FOR I=1 TO 150
40  A%(I)=I
50  NEXT I
60  ERASE A%
70  DIM A%(1000)
80  FOR I=1 TO 1000
90  A%(I)=I
100 NEXT I
110 PRINT A%
```



PUNTOS A RECORDAR

A una variable simple con el mismo nombre de la matriz no le afecta esta instrucción, como puedes ver en el programa anterior.

La instrucción ERASE sólo necesita el nombre de la matriz para borrarlo; es decir, no debes escribir ERASE A%(150), ya que será suficiente escribir ERASE A%.

PRECAUCIONES

Si intentas borrar una matriz inexistente, obtendrás un error de llamada a una función ilegal *(Illegal function call)*. En cambio, si intentas redimensionar una matriz que no había sido borrada previamente tendrás un error en el redimensionamiento de una matriz.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DIM

Parte primera, tema 12: "Lectura de datos de matrices".

ERL Línea ERrónea

DESCRIPCION

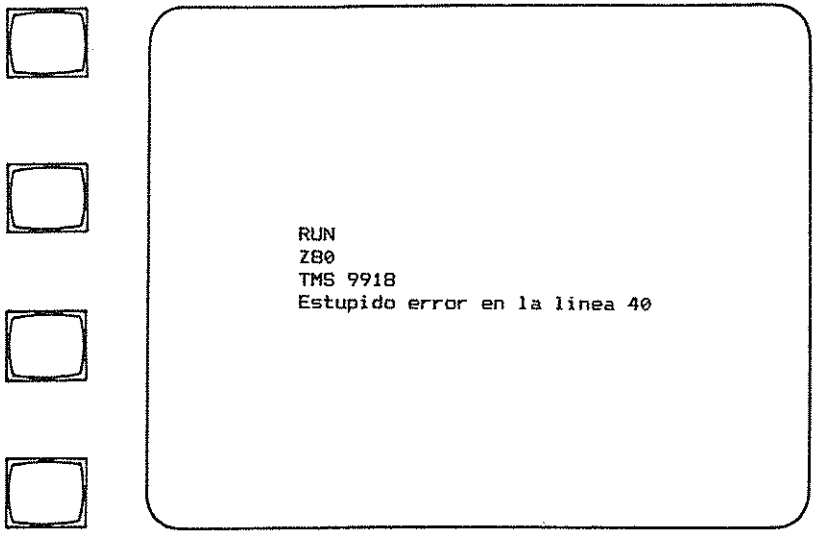
ERL es una variable reservada del sistema, que contiene el número de línea en que se ha producido un error. Se suele emplear en rutinas de depuración de errores.

SINTAXIS

<var-num> = ERL
PRINT ERL

EJEMPLO

```
10   ON ERROR GOTO 1000
20   PRINT "Z80"
30   PRINT "TMS 9918"
40   PRINT "ERROR"
50   END
1000 E%=ERL
1010 PRINT "Estupido error en la linea";E%
1020 END
```



PUNTOS A RECORDAR

Si el error se comete en modo directo, es decir, ejecutando una línea sin numerar directamente, la variable ERL contendrá el valor 65535.

La variable ERL es reservada; por tanto, no puedes asignarle ningún valor específico.

PRECAUCIONES

Si empleas la variable ERL en las subrutinas de depuración de errores te ayudará a encontrar las líneas donde exista algún problema.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON ERROR GOTO
ERROR
ERR

Parte segunda, tema 38: "Tratamiento de errores".

ERR Código de ERRor

DESCRIPCION

Es una variable reservada cuyo contenido es el código numérico del error que se ha cometido en el programa. Suele emplearse en las subrutinas de depuración de errores.

Toma valores entre 1 y 255 y se puede emplear en combinación con mensajes de error definidos por el propio usuario.

La variable ERR se emplea, también, para la creación de nuevos errores del usuario mediante la instrucción ERROR (véase el ejemplo siguiente).

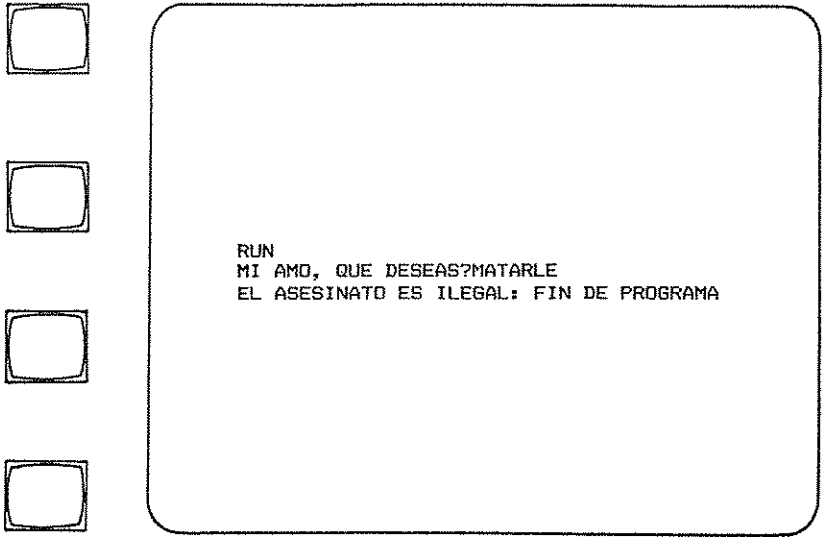
SINTAXIS

<var-num> =ERR

EJEMPLO

En esta rutina todas las órdenes que lleven la palabra MATAR al principio, al final o entre medias, son tratados como un error (255) y se imprime un mensaje de acuerdo con ello, mediante la variable ERR.

```
10   ON ERROR GOTO 1000
20   INPUT "MI AMO, QUE DESEAS";A$
30   IF INSTR(A$,"MATAR") THEN ERROR 255
40   PRINT "VALE" : END
1000 IF ERR=255 THEN PRINT "EL ASESINATO ES ILEGAL
: FIN DE PROGRAMA" : END
1010 END
```



```
RUN
MI AMO, QUE DESEAS?MATARLE
EL ASESINATO ES ILEGAL: FIN DE PROGRAMA
```


PUNTOS A RECORDAR

A la variable ERR no se le puede asignar ningún valor, ya que es una variable reservada.

PRECAUCIONES

Te recomendamos que la emplees en la detección de los errores de tu programa.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ERL
ERROR
ON ERROR GOTO
RESUME

Parte segunda, tema 38: "Tratamiento de errores".

ERROR

DESCRIPCION

Hay, básicamente, dos formas de emplear la instrucción ERROR:

1) Para simular errores.
2) Para crear tus propios errores empleando la instrucción ON ERROR GOTO.

Una instrucción ERROR con un argumento entero hará que el ordenador crea que hay un error, e imprimirá el número de línea y el mensaje de error correspondiente al argumento.

Una instrucción ERROR, conjuntamente con el uso de la sentencia de detección de errores ON ERROR GOTO, te permitirá crear tus propios errores.

El MSX tiene 36 errores estándar, con códigos entre 1 y 60; por tanto, puedes emplear los códigos de error entre 61 y 255 como errores especiales de tu propia creación.

Para programar tus propios errores lo primero que has de hacer es situar la instrucción ON ERROR GOTO <línea> al principio del programa, situando el ordenador en modo de captura de errores. Luego, si en el programa se cumple alguna condición que tú consideres errónea, llama a la rutina de errores. Esto lo puedes hacer así:

IF <condición> THEN ERROR <código de error>

Entonces activarás la rutina que tú hiciste, donde se tratará el error. En esta subrutina debes tener instrucciones como ésta:

IF ERR = <código-error> THEN PRINT "Mensaje de error"

La rutina de errores puede finalizar la ejecución, o bien continuarla en una determinada línea, según emplees las instrucciones END o RESUME.

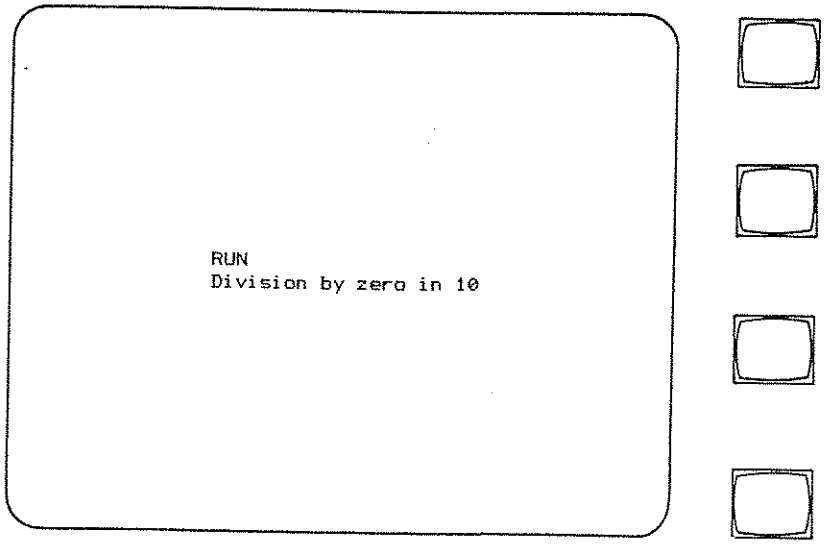
SINTAXIS

ERROR <código de error>
Siendo <código de error> un entero comprendido entre 0 y 255.

EJEMPLOS

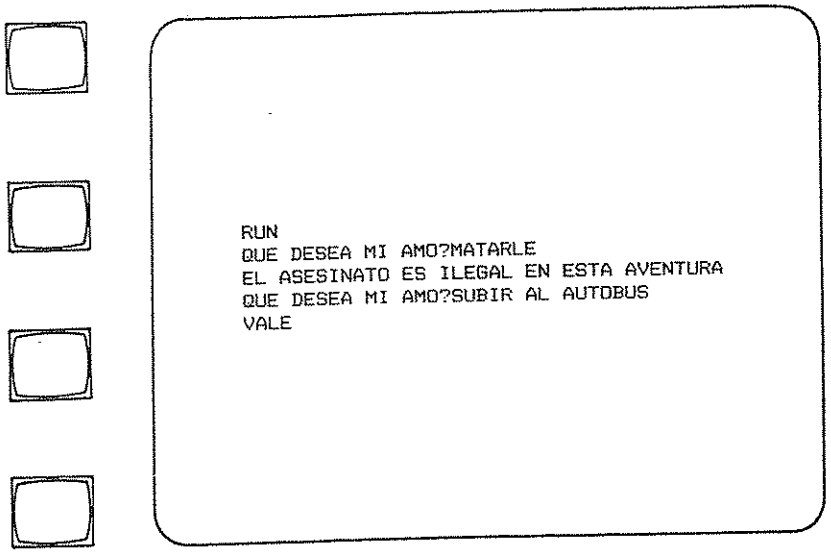
1. Aquí tienes un programa corto que te simula un error.

```
10 ERROR 11
```

2. En esta rutina todas las órdenes que lleven la palabra MATAR al principio, al final o entre medias, se tratarán como un error (255) y se imprimirá un mensaje en la rutina de tratamiento de errores valiéndose de la variable ERR.

```
10   ON ERROR GOTO 1000
20   INPUT "QUE DESEA MI AMO";A$
30   IF INSTR(A$,"MATAR")THEN ERROR 255
40   PRINT "VALE":END
..
...
1000 IF ERR=255 THEN PRINT"EL ASESINATO
ES ILEGAL EN ESTA AVENTURA":RESUME 20
1010 END
```



PUNTOS A RECORDAR

Te aconsejamos que cuando definas tus propios errores les asignes códigos de error, comenzando desde 255 hacia atrás, para evitar problemas con posibles futuros cambios en los ordenadores MSX.

Si a un código de error no se le ha asignado un mensaje, el ordenador mostrará en pantalla "*Unprintable error*" (error sin mensaje) y detendrá la ejecución del programa.

La rutina de detección de errores no ha de tener ninguno de éstos; en caso de que lo tuviera se detectaría durante la ejecución, daría el mensaje de error correspondiente y detendría la ejecución.

Una vez que la rutina esté correcta el ordenador detectará también los errores en modo directo.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON ERROR GOTO
ERL
ERR
RESUME

Parte segunda, tema 38: "Tratamiento de errores".

EXP

DESCRIPCION

Esta función nos proporciona el valor del número *e* elevado al argumento dado (*x*).

e^X

SINTAXIS

EXP(<const-num>)
EXP(<var-num>)
EXP(<exp-num>)

EJEMPLO

```
PRINT EXP(1)
2.7182818284588
```

PUNTOS A RECORDAR

El resultado de esta función se encuentra definido en doble precisión.

PRECAUCIONES

Si el exponente es tan grande que dificulta la operación al ordenador, se producirá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

LOG

Parte primera, tema 19: "Funciones matemáticas".

FIX

DESCRIPCION

Esta función nos devuelve como resultado la parte entera del argumento dado. La parte decimal del argumento se trunca.

La función FIX equivale a SGN(X)*INT(ABS(X)).

Esta función da el número entero inmediatamente inferior para números positivos, y el número entero inmediatamente superior para números negativos.

```
FIX(1.87) = 1
FIX(-12.88) = -12
```

SINTAXIS

```
FIX(<const-num>)
FIX(<var-num>)
FIX(<exp-num>)
```

EJEMPLOS

```
PRINT FIX(-1.2209)
```

aparecerá en pantalla el valor –1, y

```
A% = FIX(10.99)
```

hará que A% tome el valor 10.

PUNTOS A RECORDAR

La función INT devuelve el número entero inmediatamente más bajo como resultado, indiferentemente de que el argumento sea positivo o negativo. Por ejemplo, INT(–1.3) = –2. Esta es la diferencia entre las funciones INT y FIX.

PRECAUCIONES

Esta es una función numérica; por tanto, no debe mezclarse con cadenas de caracteres. De ser así te producirá un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INT

Parte primera, tema 15: "Funciones".

FOR bucle FOR/TO/NEXT

DESCRIPCION

Es una de las instrucciones empleadas en los bucles. En estos bucles se conoce previamente el número de veces que se ha de repetir. STEP es otra instrucción de estos bucles, la cual define la cantidad en que irá incrementado el índice contador en cada pasada.

Por ejemplo:

```
10 FOR I = 1 TO 3
20 PRINT I
30 NEXT I
```

nos mostrará en la pantalla los números 1, 2 y 3.

El índice contador I toma inicialmente el valor 1 y ejecuta las instrucciones que se encuentren entre FOR y NEXT, en este caso PRINT I. Una vez que llega a NEXT se incrementará el contador la cantidad fijada por STEP; aquí es 1 al haber efectuado omisión de dicha instrucción, y se compara con el límite dado. En el caso de ser mayor no se ejecuta el bucle, y se continúa con la instrucción siguiente a NEXT.

La indicación STEP ha de ir en la misma línea que contiene el FOR. Veamos el ejemplo anterior modificado:

```
10 FOR I = 1 TO 3 STEP 0.5
20 PRINT I
30 NEXT I
```

que nos dará los números 1, 1. 5, 2, 2. 5 y 3 como resultado.

De este modo podemos incluir bucles uno dentro de otro. Observa este otro ejemplo:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 5
30 PRINT I,J
40 NEXT J
50 NEXT I
```

y el resultado será: 1, 1,2, 1,3, etc.

Aquí tienes unas reglas para bucles imbricados:

1) Cada bucle ha de tener un contador diferente.
2) La sección NEXT del bucle más interno se ha de ejecutar antes que la del más externo.
3) Puedes crear tantos bucles anidados como quieras, unos detrás de otros. La única limitación es la memoria de tu ordenador.

4) Una única sentencia NEXT con la lista de todas las variables contadoras de todos los bucles puede sustituir a toda la lista de NEXT seguidos de sus variables.

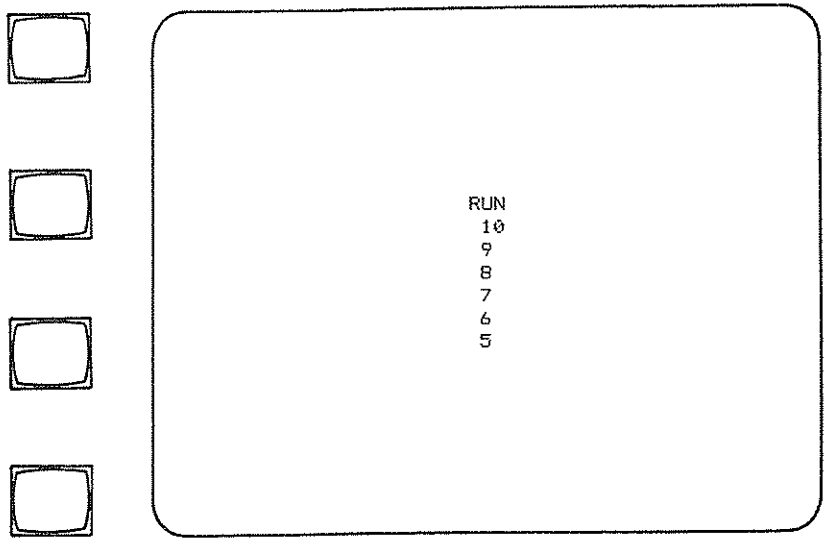
SINTAXIS

FOR <var-num> = <número> TO <número>
FOR <var-num> = <número> TO <número> STEP <número>

EJEMPLO

También puedes crear estos bucles con un incremento negativo, como en el siguiente ejemplo:

```
10 FOR F=10 TO 5 STEP -1
20 PRINT F
30 NEXT F
```



PUNTOS A RECORDAR

Si estando en un bucle te sales efectuando un salto, has de volver a él de nuevo. Por tanto, puedes emplear la instrucción GOSUB, que retornará al punto posterior al que saltó, al encontrar RETURN. Los saltos mediante la instrucción GOTO desde un bucle no son recomendables: son de un mal estilo de programación.

PRECAUCIONES

Cada instrucción FOR ha de tener su NEXT correspondiente y viceversa; de no ser así se producirá un error de NEXT sin FOR correspondiente *(NEXT without FOR)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

NEXT
STEP
TO

Parte primera, tema 5: "Cómo emplear los bucles".

FRE Cantidad de memoria libre *(amount of memory FREe)*

DESCRIPCION

Esta variable del sistema te indica la cantidad de memoria disponible en el área de programas BASIC y en el área de cadenas de caracteres.

SINTAXIS

FRE(0) te da la cantidad de memoria disponible para tu programa en BASIC.

FRE("") te da la cantidad de memoria disponible para almacenar cadenas de caracteres.

EJEMPLOS

```
PRINT FRE(0)
28815
PRINT FRE ("")
200
```

PUNTOS A RECORDAR

FRE(0) tiene el mismo valor que FREE en el mapa de memoria.

PRECAUCIONES

Necesitas los argumentos (0) y ("") para trabajar con la función FREE.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

Parte segunda, tema 48: "Mapa de memoria".

GOSUB salta a la subrutina *(GO to SUBrutine)*

DESCRIPCION

Es muy frecuente que en un programa BASIC ejecutes las mismas instrucciones en ciertos puntos del mismo. Para ello se te facilita la posibilidad de escribir estas instrucciones (subrutina) en cualquier parte del programa y llamarlas, mediante GOSUB, desde donde las necesites.

Cuando el intérprete BASIC encuentre una llamada GOSUB irá a la línea indicada y continuará ahí su ejecución hasta encontrar la palabra RETURN, con la que volverá al punto desde donde se hizo la llamada.

Puedes llamar a una subrutina desde otra; así podrás tener tantas subrutinas encadenadas como desees mientras no desbordes la memoria. Incluso te será posible tener una subrutina recursiva que se llame a sí misma.

SINTAXIS

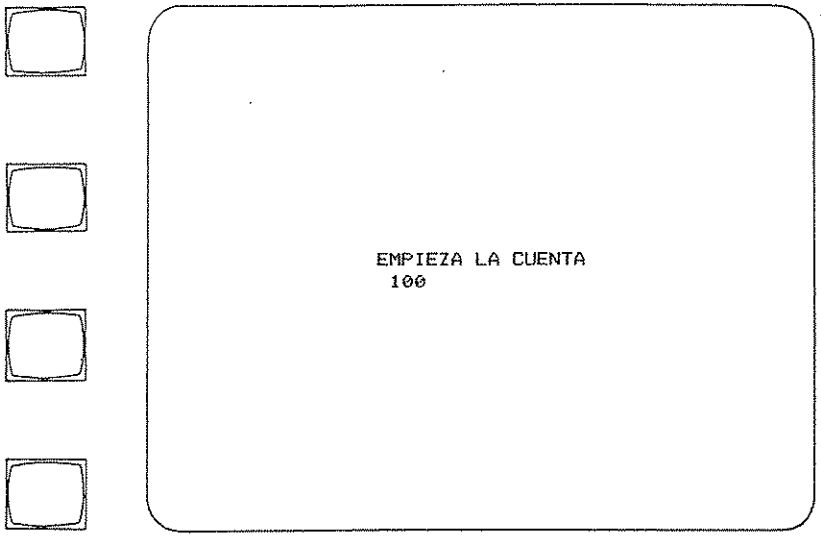
GOSUB <línea>

EJEMPLO

Aquí tienes un ejemplo de una subrutina recursiva; ésta se llama a sí misma en la línea 110 hasta que satisface la condición

```
10 PRINT"EMPIEZA LA CUENTA"
20 A=1
30 GOSUB 100
40 PRINT A
50 END
90 REM Subrutina
100 A=A+1
110 IF A<>100 THEN GOSUB 100
120 RETURN
```

Leccion 12



PUNTOS A RECORDAR

No es conveniente que los saltos a la subrutina tengan un comentario (REM) en la línea primera, ya que en un futuro quizá necesites borrarlos para ahorrar memoria.

El número de la línea referenciado por la llamada no puede ser una expresión variable. GOSUB A%*10 es incorrecto.

Una práctica muy normal en programación es la de distinguir las subrutinas; esto lo puedes hacer mediante comentarios (REM) que indiquen qué es lo que hace la subrutina.

En una subrutina es posible tener varios puntos de retorno de la llamada (RETURN), según te sea necesario.

Como los programas se ejecutan secuencialmente, asegúrate de poner un END antes de la parte de las subrutinas para que éstas no se ejecuten en un momento no necesario.

La instrucción GOSUB se usa ampliamente con las instrucciones de detección, tales como ON SPRITE, etc. Busca la parte correspondiente para obtener más detalles.

PRECAUCIONES

Un error de línea no existente lo producirá una instrucción GOSUB <línea>, en caso de no estar definida la línea a la que apunta.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

RETURN
ON ... GOSUB

ON INTERVAL GOSUB
ON KEY ... GOSUB
ON SPRITE GOSUB
ON STOP GOSUB
ON STRIG GOSUB

Parte primera, tema 17: "Estructura de tus programas".

GOTO salta a la línea *(GOTO line number)*

DESCRIPCION

Esta instrucción indica al ordenador que vaya a la línea indicada y continúe la ejecución desde esa citada línea.

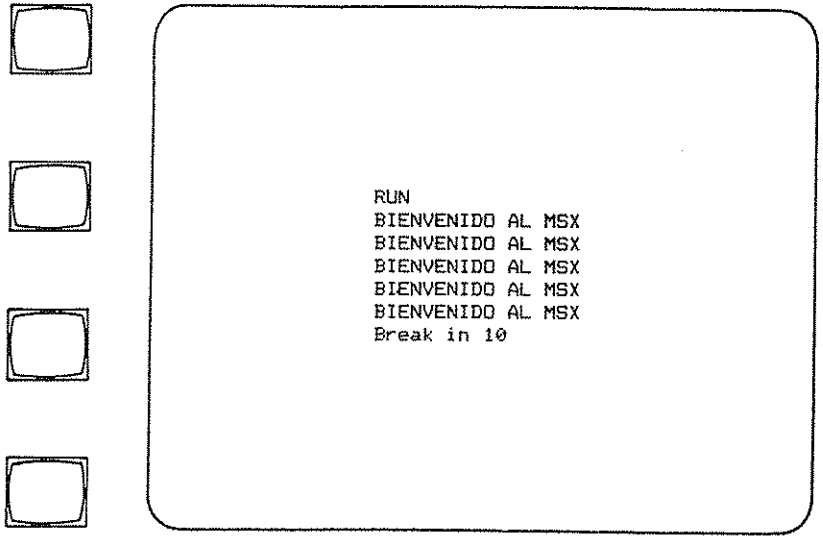
En las instrucciones del tipo IF/THEN/GOTO puedes sustituir la parte THEN GOTO por THEN o GOTO simplemente; tu ordenador lo entenderá perfectamente.

SINTAXIS

GOTO <línea>

EJEMPLO

```
10 PRINT "BIENVENIDO AL MSX"
20 GOTO 10
```



PUNTOS A RECORDAR

La línea a que se salta con la instrucción GOTO puede ser la misma que la contiene. Esto se utiliza para preguntar continuamente por ciertas condiciones, y también para mantener pantallas gráficas. Por ejemplo:

```
100 IF INKEY$<>"A" GOTO 100
```


PRECAUCIONES

Si la instrucción GOTO <línea> apunta a una línea no existente te resultará un error de línea no definida *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON ERROR GOTO
ON ... GOTO

Parte primera, tema 3: "Escritura de un programa".

HEX$ cadena en HEXadecimal *(HEX string)*

DESCRIPCION

Esta es una función que devuelve el valor hexadecimal del argumento entero dado; el resultado lo devuelve en forma de cadena de caracteres. Por ejemplo, HEX$(255) dará como resultado la cadena "FF".

SINTAXIS

HEX$ (<número-entero>)

El rango de empleo de la función va desde −32768 a 65535.
Los números negativos se representan en complemento a 2; o sea:

HEX$(65535−X) = HEX$(−X).

EJEMPLOS

```
PRINT HEX$(255)
A$=HEX$(NUM%)
```

PUNTOS A RECORDAR

&H<número> nos da el valor decimal de un número hexadecimal.

PRECAUCIONES

Recuerda que el argumento de esta función ha de ser un número entero, mientras que el resultado es una cadena de caracteres.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OCT$
BIN$

Parte segunda, tema 33: "Presentación de los sistemas de numeraciones del MSX".

IF

DESCRIPCION

La instrucción IF pregunta por una condición o por una combinación de ellas actuando posteriormente de acuerdo con la respuesta.
La forma de actuar de la instrucción IF es:

1) Si (IF) la condición no se verifica, entonces continuar la ejecución en la línea siguiente.
2) ELSE es parte de la instrucción IF/THEN/ELSE. Indica al ordenador que si no se verifica la condición salte las instrucciones que siguen a THEN y ejecute las que siguen a ELSE.

Puedes escribir tantas instrucciones IF/THEN/ELSE en una línea como desees; incluso puedes anidarlas. La única limitación que esto tiene son los 255 caracteres que pueden ir en una línea. En caso de que hubiera menos ELSE que THEN, aquéllos irían agrupados con el THEN anterior más próximo.

```
IF THEN (IF THEN ELSE)
IF THEN (IF THEN (IF THEN ELSE) ELSE) ELSE
```

Cuando la instrucción es de la forma IF ... THEN GOTO <línea> se puede simplificar de estas dos maneras:

1) IF ... THEN <línea>
2) IF ... GOTO <línea>

Las sentencias ELSE GOTO <línea> se simplifican sustituyéndolas por ELSE <línea>.

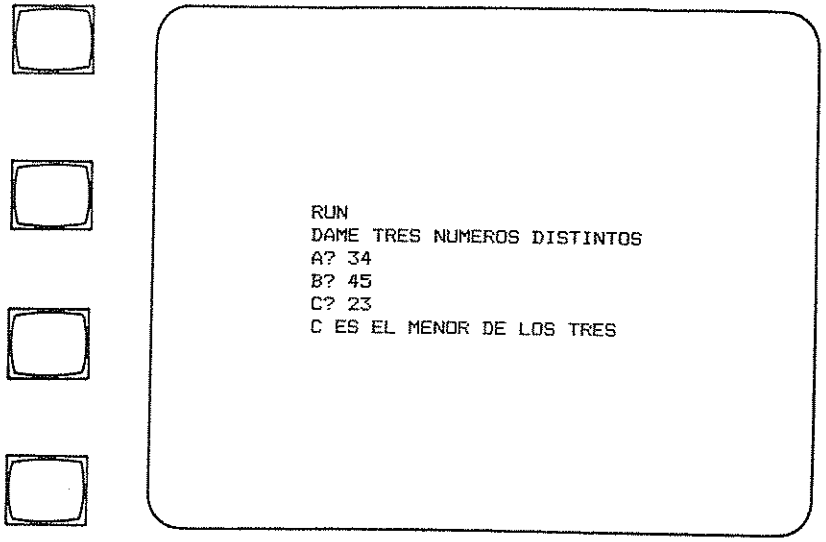
SINTAXIS

```
IF <condición> THEN <sentencias>
IF <condición> THEN <línea>
               GOTO
IF <condición> THEN <sentencias o línea> ELSE <sentencias o línea>
               GOTO                      GOTO
```

EJEMPLOS

En la línea 50 tienes una sentencia IF THEN simple.
En la línea 60 tienes la estructura IF/THEN/ELSE IF/THEN/ELSE.

```
10 PRINT"DAME TRES NUMEROS DISTINTOS"
20 INPUT "A";A%
30 INPUT "B";B%
40 INPUT "C";C%
50 IF A%=B% OR A%=C% OR B%=C% THEN 10
60 IF A%<B% AND A%<C% THEN PRINT "A" ELSE IF
B%<A% AND B%<C% THEN PRINT "B" ELSE PRINT "C"
80 PRINT "ES EL MENOR DE LOS TRES"
```



```
RUN
DAME TRES NUMEROS DISTINTOS
A? 34
B? 45
C? 23
C ES EL MENOR DE LOS TRES
```

PUNTOS A RECORDAR

Las instrucciones IF/THEN/ELSE llegan a ser demasiado largas para que quepan en una línea, ya que son múltiples las instrucciones que pueden ir después de THEN y del ELSE. Una buena solución en estos casos es el empleo de subrutinas, llamándolas mediante GOSUB.

Cuando programes con instrucciones IF/THEN/ELSE anidadas, ten cuidado: tus programas pueden llegar a ser liosos y nada consistentes.

PRECAUCIONES

Los posibles problemas que te causen las sentencias IF/THEN/ELSE suelen estar en la parte de la <condición>.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

THEN
ELSE
AND
OR
NOT

Parte primera, tema 6: "El uso de condiciones".

Parte segunda, tema 35: "Algebra de Boole (II): la instrucción IF/THEN/ELSE".

IMP IMPlica

DESCRIPCION

Es una de las operaciones lógicas que tiene el MSX.
Los resultados que produce se representan en la tabla de verdad siguiente:

IMP	X	Y	X IMP Y
	0	0	1
	1	0	0
	0	1	1
	1	1	1

100 IMP 50 se calculará del siguiente modo:

	X	100	0000000001100100
IMP	Y	50	0000000000110010
		−69	1111111110111011

SINTAXIS

<var-num> = <const-num> IMP <const-num>
<var-num> = <var-num> IMP <exp-num>

y así todas las combinaciones numéricas posibles.

EJEMPLO

```
10 S=99
20 PRINT S IMP S
30 PRINT (-S IMP S)
40 PRINT (S IMP -S)
```



```
RUN
-1
 99
-99
```

PUNTOS A RECORDAR

Todas estas relaciones se verifican con este operador:

X IMP X = −1
−X IMP X = X
X IMP −X = −X

IMP opera con argumentos de 16 bits.

PRECAUCIONES

Si uno de los argumentos se sale del rango de los números enteros te causará un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

AND
OR
NOT
XOR
EQV

Parte segunda, tema 34: "Algebra de Boole (I): operadores lógicos".

INKEY$

DESCRIPCION

Esta función comprueba si en ese momento se está pulsando alguna tecla; en caso de que no se esté pulsando ninguna tecla su resultado será una cadena de caracteres vacía ("").

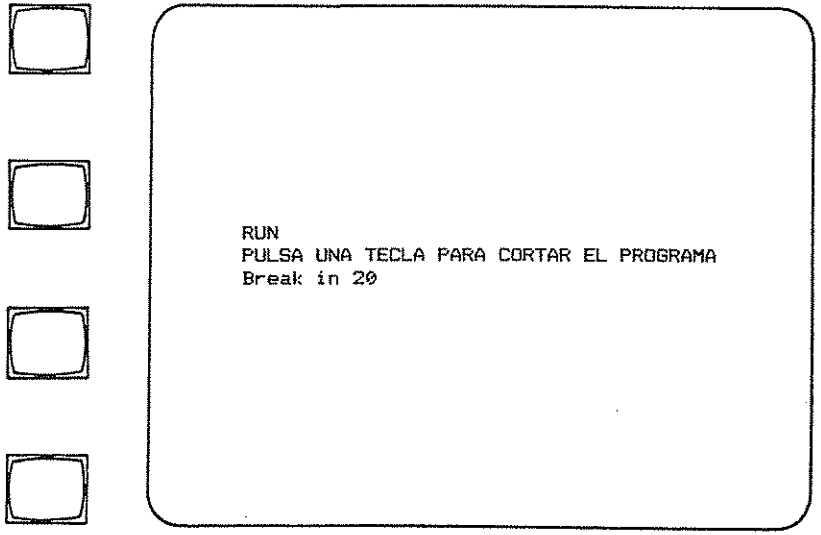
SINTAXIS

<cad-var> = INKEY$

EJEMPLO

Este programa demuestra cómo hacer esperar el ordenador hasta que pulses una tecla. En la línea 20 se pregunta por la tecla pulsada y se guarda en A$. Si no se ha pulsado ninguna tecla se indica al ordenador que vuelva a la misma línea, y así sucesivamente hasta que pulses una tecla.

```
10 PRINT "PULSA UNA TECLA PARA CORTAR EL PROG
RAMA"
20 A$=INKEY$ :IF A$="" THEN 20 ELSE STOP
```



```
RUN
PULSA UNA TECLA PARA CORTAR EL PROGRAMA
Break in 20
```


PUNTOS A RECORDAR

Aquí tienes las diferencias entre INPUT e INKEY$:

1) INKEY$ no visualiza la interrogación "?" en la pantalla.
2) INKEY$ no espera a que pulses una tecla. Si no estás pulsando una tecla cuando se ejecute la instrucción INKEY$ el resultado será una cadena de caracteres nula.
3) INKEY$, al revés que INPUT, no muestra en pantalla el carácter leído.
4) La función INKEY$ reconoce las teclas TAB y DEL .
5) INKEY$ no hace como INPUT, que te pasa al modo texto si estuvieras en modo gráfico.
6) INKEY$ sólo devuelve un carácter; sin embargo, INPUT puede devolver hasta una línea de caracteres.

PRECAUCIONES

Si igualas INKEY$ a una variable de tipo numérica te causará un error en el tipo de datos (*Type mismatch error*).

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INPUT$

Parte primera, tema 10: "Programación interactiva".

INP entrada de puerto *(IN from Port)*

DESCRIPCION

Esta función te devuelve el byte leído en el puerto que le hayas especificado. El número de puerto debe estar comprendido entre 0 y 255. El MSX no usa puertos con índice mayor que 255.

Si estás haciendo programas comerciales procura no usar esta sentencia. Emplea las BIOS, será más sencillo.

SINTAXIS

<var-num> =INP(<puerto-num>)

EJEMPLO

```
PRINT INP(1)
```

PUNTOS A RECORDAR

INP realiza la función contraria a la instrucción OUT.

PRECAUCIONES

Si no conoces muy bien su manejo, procura no emplearla.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OUT
WAIT

INPUT

DESCRIPCION

Esta instrucción hace interactivos tus programas, permitiendo que des valores a variables numéricas y de tipo cadena mientras el programa se está ejecutando.

La instrucción INPUT detiene el programa y espera que asignes un valor a través del teclado a las variables especificadas.

Puedes incluir un mensaje aclarativo delante de la interrogación que aparece en la pantalla al ejecutarse un INPUT.

En una misma instrucción INPUT puedes pedir todos los valores de variables que quieras, que han de ir configurando una lista separadas por comas; teniendo en cuenta que el tipo de dato y el tipo de la variable ha de ser el mismo, en caso de tener un fallo te aparecerá un mensaje como éste:

```
?Redo from start
```

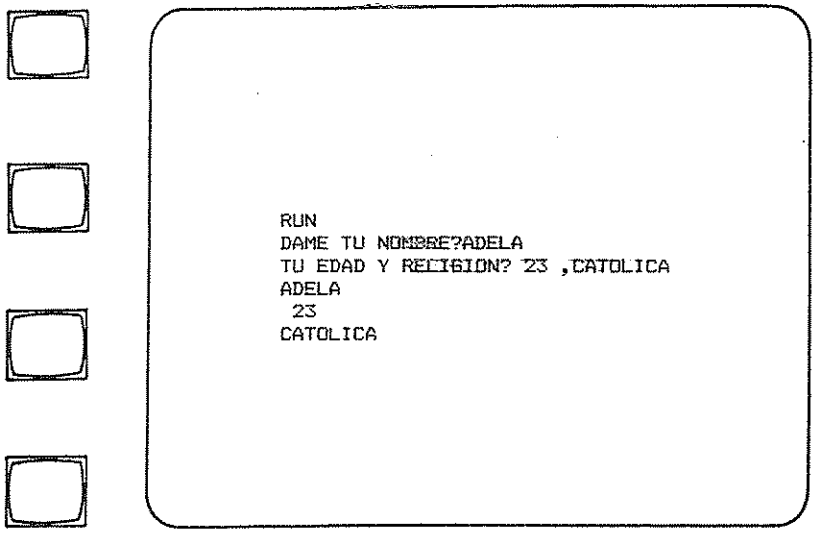
y tendrás que volver a dar valores a todas las variables de la sentencia INPUT desde la primera.

SINTAXIS

INPUT <lista de variables numéricas o de tipo cadena de caracteres separadas por comas>
INPUT "mensaje";<lista de variables>

EJEMPLO

```
10 INPUT "DAME TU NOMBRE";N$
20 INPUT "TU EDAD Y RELIGION";E%,R$
30 PRINT N$
40 PRINT E%
50 PRINT R$
```



PUNTOS A RECORDAR

La instrucción INPUT te pasará al modo texto en caso de que se ejecutara estando en modo gráfico.

La instrucción INPUT te responderá de estos tres modos en caso de que hubiera un error:

1) En el tipo de datos el mensaje será:

   ?Redo from start

y tendrás que volver a empezar (desde la primera variable pedida).

2) Por demasiadas instrucciones INPUT, el mensaje será:

   ?Extra ignored

omite los INPUT excedentes y continúa la ejecución del programa.

3) Si no fueran suficientes los valores entrados, el mensaje será:

   ??

es decir, dos interrogaciones en espera de los valores restantes que faltaban.

El único modo de omitir la instrucción INPUT es haciendo <CTRL><C> o <CTRL><STOP>. Si has omitido un INPUT tienes la posibilidad de volver a la ejecución en ese punto mediante el comando CONT, pero tendrás que dar los valores de las variables pedidas en la instrucción INPUT omitida.

Si pulsas exclusivamente la tecla <RETURN> cuando te piden el valor de una variable, ésta seguirá teniendo el mismo valor que tenía anteriormente a la instrucción INPUT.

Así como la lista de variables pedidas ha de ir separada por comas, también irán separados por comas los valores que asignas a las variables de esa lista.

PRECAUCIONES

Todos los errores que cometas al teclear los valores que te pidan serán tratados como te indicamos en el punto anterior.

Si la longitud de una cadena de caracteres pedida supera los 200 caracteres habrá un error de exceso de caracteres, a menos que aumentes la longitud de trabajo de las cadenas de caracteres mediante la instrucción CLEAR.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INKEY$
INPUT#
INPUT$
LINE INPUT

Parte primera, tema 2: "El modo directo".
Parte primera, tema 10: "Programación interactiva".

INPUT

DESCRIPCION

Esta instrucción te permite leer datos de otros dispositivos que no sean el teclado, como, por ejemplo, el cassette.

Antes de ejecutar esta instrucción tiene que estar dispuesto un canal de E/S, lo cual se hará ejecutando la instrucción OPEN, que nos indicará el número de fichero donde se encuentran los datos.

El formato de esta instrucción es idéntico al de la referida al teclado.

Cuando leas valores de variables numéricas se ignoran los espacios en blanco previos, las marcas de fin de línea y las líneas en blanco. Con las variables de tipo cadena de caracteres ocurre lo mismo; en este caso también se ignoran las comillas (").

SINTAXIS

INPUT # <número-fichero>,<listado de variables>

EJEMPLO

```
INPUT # 1,A$
```

PUNTOS A RECORDAR

Para abrir un fichero en cassette ejecuta:

```
OPEN"CAS:" FOR INPUT AS # 1
```

PRECAUCIONES

Si haces referencia a un fichero que no esté abierto (OPEN) producirás un error en el número de fichero *(Bad file number error).*

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OPEN
CLOSE
INPUT
INPUT$

Parte segunda, tema 47: "Manejo de ficheros".

INPUT$

DESCRIPCION

Esta instrucción devuelve un número determinado de caracteres leídos del teclado, pero ninguno de ellos se visualiza por la pantalla. La ejecución de esta instrucción finaliza con <CTRL><C>.

SINTAXIS

<var-cad> = INPUT$(<const-num>)

EJEMPLO

```
A$ = INPUT$(5)
```

PUNTOS A RECORDAR

Al revés que la instrucción INKEY$, ésta espera a que todos los caracteres pedidos se hayan leído. Por tanto, puede manejar más de un carácter cada vez.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INKEY$
INPUT$(#)
INPUT

Parte primera, tema 10: "Programación interactiva".

INPUT$ (#)

DESCRIPCION

Esta instrucción lee una cadena de caracteres, de una determinada longitud, de cualquier dispositivo que no sea el teclado; con esta instrucción no se visualiza ninguno de los caracteres leídos. Un parámetro a pasar a esta función es el número del fichero a leer, que previamente habrá sido abierto mediante la instrucción OPEN.

SINTAXIS

<var-cad> = INPUT$(<const-num>,<número-fichero>)
<var-cad> = INPUT$(<const-num>,# <número-fichero>)

EJEMPLO

W$ = INPUT$(5,#1)

PUNTOS A RECORDAR

Para abrir un fichero en el cassette ejecuta:

OPEN "CAS:" FOR INPUT AS#1

PRECAUCIONES

Si la instrucción hace referencia a un fichero que no ha sido abierto se producirá un error de número de fichero *(Bad file name error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OPEN
CLOSE
INPUT
INPUT#
INPUT$

Parte segunda, tema 47: "Manejo de ficheros".

INSTR

DESCRIPCION

Esta función analiza si una determinada cadena de caracteres está contenida en otra, y devuelve la posición en que se encuentra la cadena buscada.

INSTR(A$,B$) ... A$ es la cadena principal en la que se buscará la cadena B$.

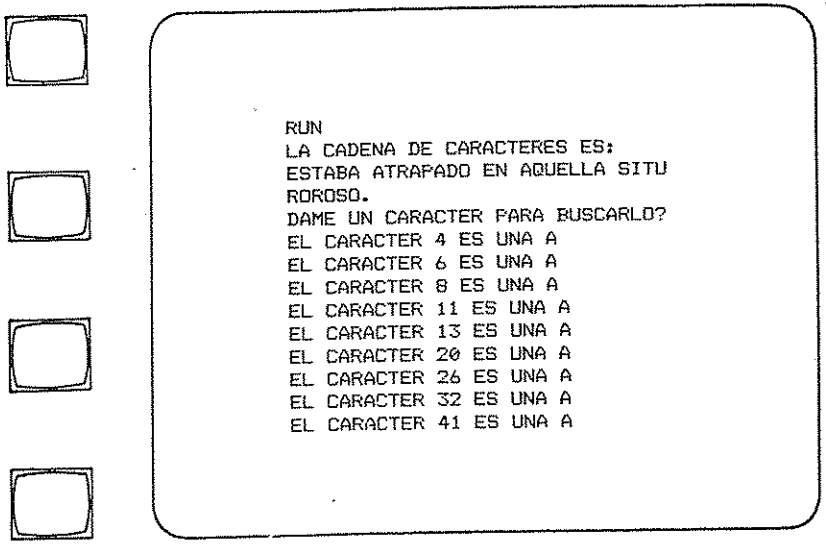
Si no se encuentra la cadena requerida el valor que devuelve la función es 0.
Si no especificas la posición de comienzo de búsqueda, entonces ésta empezará desde el primer carácter de la cadena principal. La posición de comienzo de búsqueda ha de ser un entero comprendido entre 1 y 255, inclusive.

SINTAXIS

<numérico> = INSTR(<cad-car>,"<cadena de caracteres>")
<numérico> = INSTR(<cad-car>,<cad-car>)
<numérico> = INSTR(<numérico>,<cad-car>,<cad-car>)

EJEMPLO

```
10 PRINT "LA CADENA DE CARACTERES ES:"
20 PRINT "ESTABA ATRAPADO EN AQUELLA SITUACIO
N, ERA HORROROSO."
30 A$="ESTABA ATRAPADO EN AQUELLA SITUACION,
ERA HORROROSO."
40 INPUT"DAME UN CARACTER PARA BUSCARLO";B$
50 C%=0
60 C%=INSTR(C%+1,A$,B$)
70 IF C%=0 THEN END
80 PRINT "EL CARACTER";C%;"ES UNA ";B$
90 GOTO 60
```



```
RUN
LA CADENA DE CARACTERES ES:
ESTABA ATRAPADO EN AQUELLA SITU
ROROSO.
DAME UN CARACTER PARA BUSCARLO?
EL CARACTER 4 ES UNA A
EL CARACTER 6 ES UNA A
EL CARACTER 8 ES UNA A
EL CARACTER 11 ES UNA A
EL CARACTER 13 ES UNA A
EL CARACTER 20 ES UNA A
EL CARACTER 26 ES UNA A
EL CARACTER 32 ES UNA A
EL CARACTER 41 ES UNA A
```

PUNTOS A RECORDAR

El valor de INSTR(Z,X$,Y$) será 0 en los siguientes casos:

1) Si Y$ no está contenida en X$.
2) Si Z es mayor que la longitud de la cadena X$.
3) Si X$ es una cadena de caracteres vacía (X$=“”).
4) Si X$ e Y$ son ambas cadenas de caracteres vacías.

Si intentas encontrar una cadena nula (“”) dentro de otra mediante la función INSTR, obtendrás 1 como resultado.

PRECAUCIONES

Un error de llamada ilegal a una función lo causará el intento de búsqueda a partir de una posición menor o igual a 0 o mayor que 255.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

LEFT$
RIGHT$
MID$
LEN

Parte primera, tema 14: “Las cadenas de caracteres”.

INT

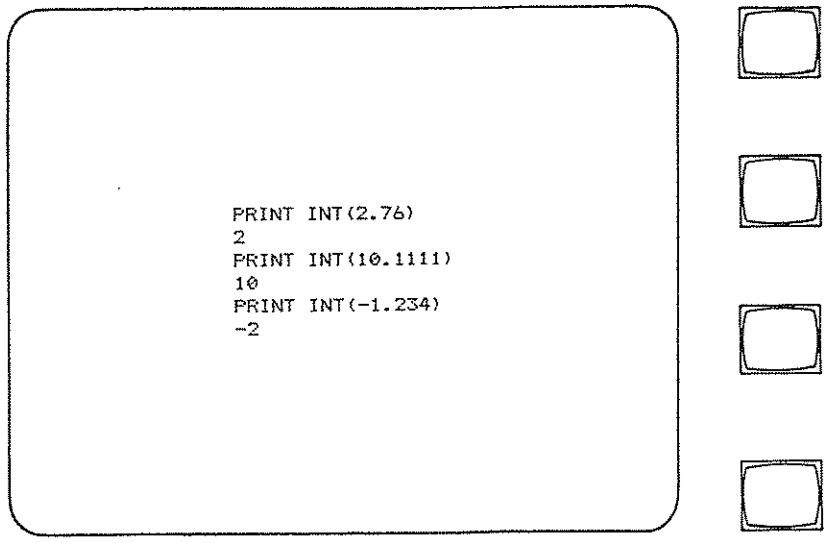
DESCRIPCION

Esta función devuelve la parte entera de un número real. Redondea al entero más pequeño y próximo al argumento, ya sea éste negativo o positivo.

SINTAXIS

INT(<const-num>)
INT(<var-num>)
INT(<exp-num>)

EJEMPLO



```
PRINT INT(2.76)
2
PRINT INT(10.1111)
10
PRINT INT(-1.234)
-2
```

PUNTOS A RECORDAR

Observa que si INT da un argumento negativo, te dará como resultado el número entero negativo menor y más próximo al argumento. Por tanto, el resultado de INT (−0,2) será −1.

La función INT es diferente de FIX; ésta devuelve el entero más pequeño y próximo para argumentos positivos, y el entero mayor y más próximo si el argumento es negativo.

PRECAUCIONES

Si intentas calcular la función INT con un argumento que sea de tipo cadena de caracteres producirás un error de tipo de datos *(Type mismatch error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

FIX

Parte primera, tema 4: "Algo más de aritmética".
Parte primera, tema 15: "Funciones".

INTERVAL ON/OFF/STOP

DESCRIPCION

El BASIC del MSX se puede programar para que ejecute determinada subrutina cada cierto intervalo de tiempo, mediante la instrucción ON INTERVAL.

INTERVAL ON/STOP/OFF activa y desactiva la subrutina de detección de los intervalos.

Para que la instrucción ON INTERVAL GOSUB se active debes ejecutar INTERVAL ON. Después de ejecutarse esta instrucción el ordenador analizará con la ejecución de cada instrucción si ha transcurrido el tiempo especificado por el intervalo para llamar a la subrutina especificadora en tu programa.

La orden INTERVAL OFF indica al ordenador que deje de analizar si ha pasado un intervalo de tiempo y desactiva la instrucción ON INTERVAL GOSUB.

INTERVAL STOP ordena que no se ejecute la subrutina indicada por ON INTERVAL GOSUB, pero que memorice los intervalos de tiempo que transcurren hasta que haya una orden INTERVAL ON, momento en que se tratarán todos los intervalos memorizados.

SINTAXIS

INTERVAL ON
INTERVAL OFF
INTERVALO STOP

EJEMPLO

Pulsa <y> para activar la rutina de detección de intervalos y <n> para desactivarla. Oirás unos sonidos *(beeps)* cada vez que transcurra un intervalo de tiempo.

```
10 ON INTERVAL=60 GOSUB 50
20 IF INKEY$="s" THEN INTERVAL ON
30 IF INKEY$="n" THEN INTERVAL OFF
40 GOTO 20
45 REM SUBRUTINA DE INTERVALOS
50 BEEP
60 RETURN
```

LINEA 10 FIJA LOS INTERVALOS A UN SEGUNDO Y APUNTA A LA SUBRUTINA (50).
LINEA 20 ACTIVA LA DETECCION DE INTERRUPCIONES A INTERVALOS DE TIEMPO.
LINEA 30 DESACTIVA ESTAS INTERRUPCIONES.
LINEA 40 BUCLE INFINITO DE SALTO A LA LINEA 20.
LINEA 60 VUELVE AL PUNTO EN DONDE DEJO EL PROGRAMA PRINCIPAL.

PUNTOS A RECORDAR

El intervalo va fijado en unidades de 1/60* de segundo.

Las interrupciones de tiempo están desactivadas en el modo directo, es decir, cuando no están ejecutando el programa, y también dentro de las rutinas de detección de errores.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON INTERVAL GOSUB.

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

* 1/50 en las versiones europeas.

KEY

DESCRIPCION

Este comando se emplea con el objetivo de definir el uso de las teclas de función. Todos los ordenadores MSX poseen cinco teclas de función; cada una de ellas tiene dos funciones asociadas (una pulsando directamente la tecla y la otra pulsando la tecla de función y <SHIFT> al mismo tiempo).

Las teclas de función poseen las siguientes funciones programadas de forma estándar:

F1 = color [e]	COLOR
F2 = auto [e]	AUTO
F3 = goto [e]	GOTO
F4 = list [e]	LIST
F5 = run [r]	Ejecuta el programa.
F6 = color 15,4,7[r]	Fija los colores del MSX.
F7 = cload"	Para cargar desde el cassette.
F8 = cont [r]	Continúa con la ejecución.
F9 = list [r] [u] [a]	Lista la última línea a que se ha hecho referencia y mueve el cursor al principio de esa línea.
F10 = [cls] run [r]	Limpia la pantalla y ejecuta el programa.

[e] = espacio en blanco
[r] = RETURN
[cls] = limpia la pantalla
[u] = mueve el cursor una línea hacia arriba.

Puedes redefinir mediante KEY cualquier tecla de función, mientras que la cadena con la que la definas no sobrepase los quince caracteres.

Si programas una función con el objetivo de que se ejecute nada más pulsar la tecla correspondiente, entonces ejecuta RETURN añadiendo CHR$(13) al final de la declaración de la función con KEY. También puedes incluir en la declaración de función la orden CLS y otros caracteres de control.

SINTAXIS

KEY <const-num>,<cadena de caracteres>
KEY <const-num>,<expresión de tipo cadena de caracteres>)

siendo <const-num> un número entero comprendido entre 1 y 10.

EJEMPLOS

KEY 2, "PRINT FRE(0)" + CHR$(13)	Te muestra el número de bytes libres por pantalla para el BASIC.

```
KEY 4, "RENUM" + CHR$(13)      Te renumera todas las líneas de
                               tu programa.
a$ = "KEY LIST"                Ejecuta la orden KEY LIST
KEY 5,a$ + CHR$(13)            <RETURN>.
```

Nota: CHR$(13) es el carácter de control correspondiente a la tecla <RETURN>.

PUNTOS A RECORDAR

Cuando conectes tu ordenador apareceran en la parte inferior de la pantalla las funciones de dichas teclas. Si no quieres que se vean basta con que des la orden KEY OFF.

Recuerda que puedes incluir cualquier carácter de control a la función definida; así conseguirás que ésta te borre la pantalla, te dé un sonido *(beep)*, etc.

PRECAUCIONES

No te olvides de entrecomillar las cadenas de caracteres de la declaración de función.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

KEY ON/OFF
KEY LIST

Parte primera, tema 8: "Las teclas de función".

KEY LIST

DESCRIPCION

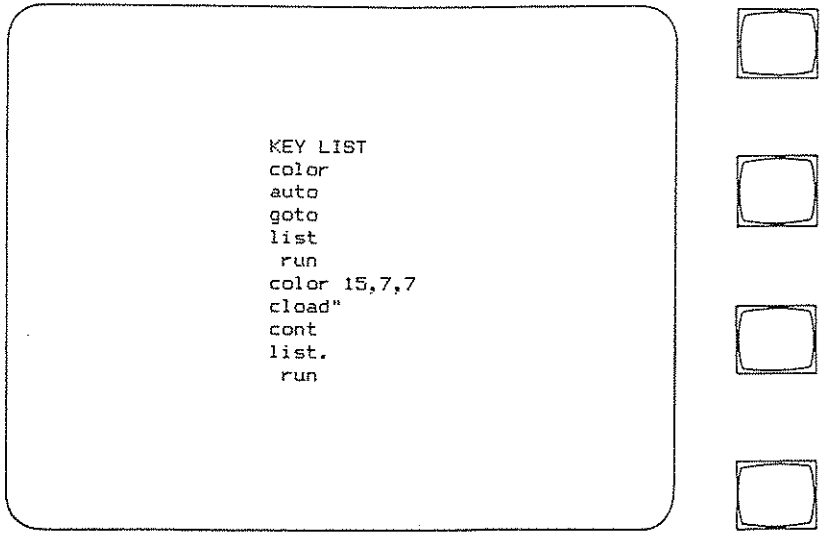
Esta instrucción te muestra todas las funciones que ejecutan las teclas de función, sin llevar a cabo ninguna de ellas.

Los caracteres de control no aparecen al listar las funciones: se transforman en espacios blancos.

SINTAXIS

KEY LIST

EJEMPLO



PUNTOS A RECORDAR

Mira la instrucción KEY para ver cómo cambiar las funciones de dichas teclas.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

KEY
KEY ON/OFF

Parte primera, tema 8: "Las teclas de función".

KEY ON/OFF

DESCRIPCION

Al conectar tu ordenador se verán en la línea 24 los contenidos de las teclas de función. Con ello puedes recordar qué es lo que hace cada una de las teclas de función, pero a veces cuando esté ejecutando tu programa te entorpece la pantalla. La instrucción KEY ON/OFF te visualiza o borra las funciones de dichas teclas en la línea 24.

SINTAXIS

KEY ON
KEY OFF

EJEMPLOS

KEY ON
KEY OFF

PUNTOS A RECORDAR

La instrucción KEY LIST te indicará en la pantalla qué hace cada una de las teclas de función.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

KEY
KEY LIST

Parte primera, tema 8: "Las teclas de función".

KEY () ON/OFF/STOP

DESCRIPCION

La instrucción KEY () ON/OFF/STOP activa/desactiva la detección de una determinada tecla de función. Si está activada la detección de una función y pulsas la tecla correspondiente se ejecutará la subrutina indicada en la instrucción ON KEY GOSUB.

Puedes activar y desactivar las teclas de función individualmente, según te convenga. Aquellas que no estén especificadas con KEY () ON estarán desactivadas.

La ejecución de KEY() STOP implica que cuando pulses la tecla de función especificada ésta se memorice, y al ejecutar KEY () ON el programa irá a la subrutina que atiende esa función, si hubiera sido pulsada.

KEY () OFF inhibirá todo tipo de interrupciones por parte de la tecla de función especificada.

SINTAXIS

KEY (<const-num>) ON
KEY (<const-num>) OFF
KEY (<const-num>) STOP

EJEMPLO

En este ejemplo, si pulsas F1 oirás dos sonidos *(beeps)* del ordenador; en cambio, pulsando F2 sólo oirás uno.

```
10 ON KEY GOSUB 50,60
20 KEY (1) ON
30 KEY (2) ON
40 GOTO 40
50 BEEP
60 BEEP
70 RETURN
```

LINEA 10 FIJA LAS SUBRUTINAS DE F1 EN LA LINEA 50 Y LA DE F2 EN LA 60.
LINEA 20 ACTIVA LA DETECCION DE INTERRUPCIONES DE F1.
LINEA 30 ACTIVA LA DETECCION DE INTERRUPCIONES DE F2.
LINEA 40 BUCLE INFINITO EN ESPERA DE UNA INTERRUPCION.
LINEA 50 SONIDO (BEEP) DE F1.
LINEA 60 SONIDO (BEEP) DE AMBAS.
LINEA 70 VUELTA AL PUNTO DESDE DONDE VINO: EN ESTE CASO LA LINEA 40.

Observa que el programa anterior no hace nada si pulsas cualquier otra tecla de función.

PUNTOS A RECORDAR

La detección de interrupciones de funciones está desactivada dentro de las subrutinas de detecciones de errores.

PRECAUCIONES

Lo primero que debes hacer es ejecutar la instrucción ON KEY GOSUB. Esta instrucción es totalmente diferente de KEY ON/OFF.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON KEY GOSUB

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

LEFT$

DESCRIPCION

La función LEFT$ devuelve los *n* caracteres más a la izquierda de la cadena pasada como argumento. Si la cadena de caracteres tiene menos de los *n* caracteres pedidos, entonces la función devuelve la cadena entera como resultado.

SINTAXIS

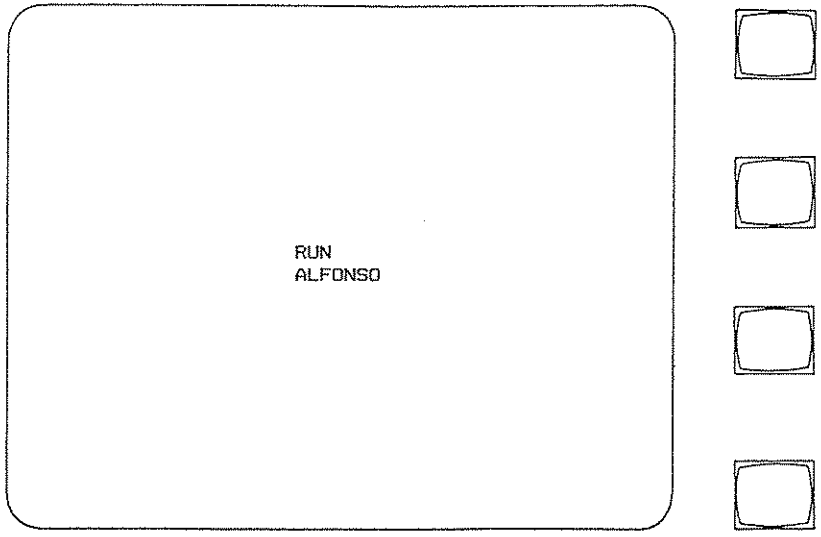
LEFT$ (<var-cad>,<const-num>)
LEFT$ (<var-cad>,<var-num>)
LEFT$ (<var-cad>,<expr-num>)

El número ha de ser un entero comprendido entre 0 y 255.

EJEMPLO

```
10 A$="ALFONSO DOMINGUEZ"
20 P=INSTR(A$," ")
30 B$=LEFT$(A$,P-1)
40 PRINT B$
```

LINEA 20 DETECTA LA POSICION EN QUE SE ENCUENTRA EL ESPACIO EN BLANCO DE A$.
LINEA 30 TOMA LOS P – 1 CARACTERES DE A$ PARA SACAR EL NOMBRE.



PUNTOS A RECORDAR

Si en el argumento se especifica 0, el resultado es una cadena de caracteres vacía ("").

Si en el ejemplo anterior quieres obtener el apellido, emplea la función MID$ como te indicamos:

```
PRINT MID$ (A$, P + 1)
```

PRECAUCIONES

Si el número del argumento no está comprendido entre 0 y 255 se producirá un error en el tipo de datos *(Type mismatch error)*.

Si no existe la variable a la que llamas con la función tendrás un error de llamada a una función ilegal *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INSTR
RIGHT$
MID$
LEN

Parte primera, tema 14: "Las cadenas de caracteres".

LEN

DESCRIPCION

Esta función devuelve la longitud de la cadena de caracteres pasado como parámetro.

También se cuentan los caracteres de control y los espacios en blanco.

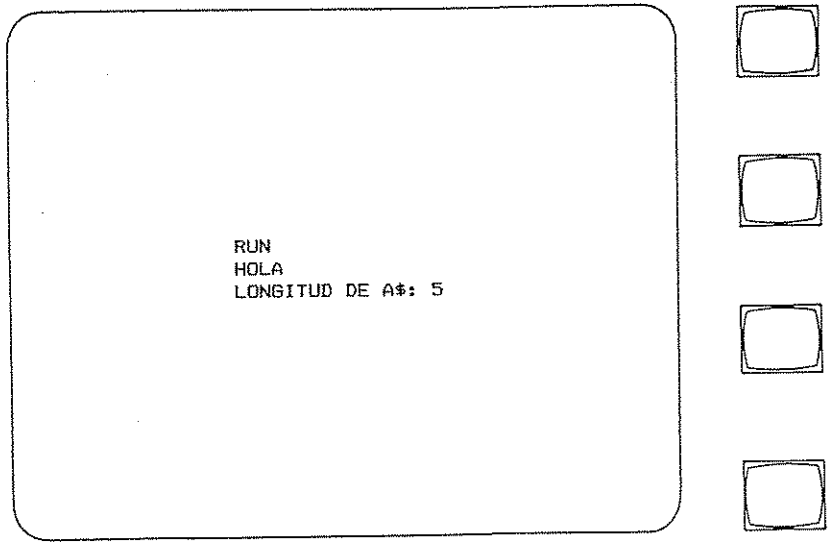
SINTAXIS

LEN(<cadena de caracteres>)

EJEMPLO

```
10 A$="HOLA"+CHR$(7)
20 B=LEN(A$)
30 PRINT A$
40 PRINT"LONGITUD DE A$:";B
```

LINEA 10 CHR$(7) ES UN SONIDO (BEEP).
LINEA 30 SACA EN LA PANTALLA HOLA Y EMITE UN SONIDO.



PUNTOS A RECORDAR

Esta función siempre te dará un resultado entero.

PRECAUCIONES

El resultado será cero si la cadena a que haces referencia no existe.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

RIGHT$
LEFT$
MID$
INSTR

Parte primera, tema 15: "Funciones".

LET

DESCRIPCION

Emplea la instrucción LET cuando desees hacer una asignación a una variable numérica o de tipo cadena de caracteres, como en este ejemplo.

```
LET A = 100
```

La instrucción LET es opcional, e incluso es mejor no usarla, ya que ocupa memoria sin ser necesaria. A = 100 equivale al ejemplo anterior.

SINTAXIS

LET <variable> = <expresión>

EJEMPLO

```
LET A=100
LET F$="FIN"+"DE"+"SEMANA"
```

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 2: "El modo directo".

LINE dibuja una LINEa

DESCRIPCION

Esta instrucción de gráficos dibuja rectas y rectángulos; éstos podrán ir coloreados.

SINTAXIS

LINE-<especificación de coordenadas>

dibuja una recta, con el último color de texto especificado al ordenador, desde el último punto referido hasta el especificado.

LINE-<especificación de coordenadas>;<color>

dibuja una recta desde el último punto referenciado por el ordenador hasta el especificado, con el color asimismo especificado.

LINE <especificación de coordenadas>-<especificación de coordenadas>

dibuja una recta entre los puntos especificados, con el último color de tinta especificado al ordenador.

LINE <especificación de coordenadas>-<especificación de coordenadas>,<color>

dibuja una recta entre los puntos especificados, con el color indicado.

LINE <especificación de coordenadas>-<especificación de coordenadas>,<color>, B

dibuja un rectángulo, siendo la primera coordenada la del vértice superior izquierdo y, la segunda, la del vértice inferior derecho, con el color especificado.

LINE <especificación de coordenadas>-<especificación de coordenadas>,<color>, BF

dibuja un rectángulo, siendo la primera coordenada el vértice superior izquierdo y, la segunda, el vértice inferior derecho, con el color especificado y posteriormente coloreándolo con dicho color.

Significado de <especificación de coordenadas>

Hay dos modos, uno relativo y el otro absoluto:

1) (<coordenada x>,<cordenada y>)

Este modo especifica un punto concreto de la pantalla. La coordenada X ha de estar entre 0 y 255, y la coordenada Y entre 0 y 191.

2) STEP (<incremento x>,<incremento y>).

Estas coordenadas aluden al último punto referido por el ordenador. Si el punto resultante se sale de la pantalla, el ordenador dibujará hasta el límite de ésta.

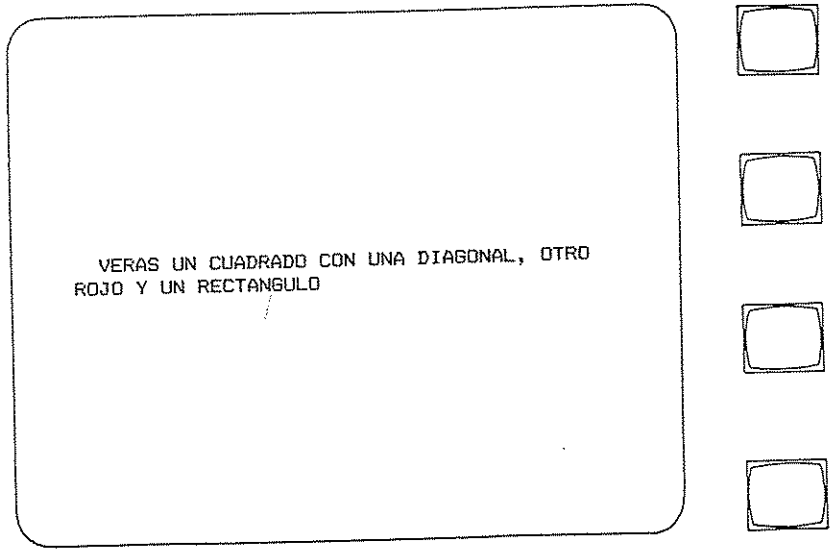
<incremento x> e <incremento y> serán positivos o negativos según la dirección en que desees dibujar.

Nota: <incremento x>, <incremento y>, <coordenada x> y <coordenada y> pueden ser variables, expresiones o constantes numéricas.

Cuando necesites dos especificaciones de coordenadas se acepta la mezcla de coordenadas relativas y absolutas mientras estén separadas por el signo menos ("–").

EJEMPLO

```
10 SCREEN 2
20 COLOR 4,1,1
30 CLS
40 LINE (50,50)-(100,100)
50 LINE (50,50)-(100,100),4,B
60 LINE STEP(-50,20)-STEP(50,50),9,BF
70 X=10:Y=10
80 LINE (15*X,10*Y)-STEP(50,20),6,B
90 GOTO 90
```



PUNTOS A RECORDAR

Las coordenadas reales son truncadas a enteros. El código de color ha de estar comprendido entre 0 y 15. (Véase COLOR.)

PRECAUCIONES

Las coordenadas pueden superar el rango permitido en la pantalla, pero si superan el rango (−32768, 32767) causarán un error de desbordamiento *(Overflow error)*. Por ejemplo: LINE (−100000, 100000) dará un error de desbordamiento.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PAINT
COLOR
DRAW

Parte primera, tema 24: "Dibuja rectas y rectángulos".
Parte segunda, tema 41: "Gráficos avanzados (II): el color en el MODO 2 de alta resolución gráfica".

LINE INPUT

DESCRIPCION

Esta instrucción es parecida a la INPUT, excepto que te permite leer sentencias enteras de hasta 200 caracteres, incluyendo las comas.

Puedes incluir mensajes, pero con esta instrucción no se imprimirá el símbolo "?", al revés que con INPUT.

SINTAXIS

LINE INPUT <car-var>
LINE INPUT "<mensaje>";<car-var>

EJEMPLO

Introduce esta rutina en tu ordenador.

```
10 LINE INPUT "Que puedo hacer ahora?";S$
20 PRINT "De acuerdo ";S$
```

```
RUN
Que puedo hacer ahora?COMER,BEBER Y DORMIR
De acuerdo COMER,BEBER Y DORMIR
```

PUNTOS A RECORDAR

Puedes salir de la ejecución de LINE INPUT con <CTRL><C> o <CTRL><STOP>. Con la orden CONT seguirás ejecutando LINE INPUT desde el principio.

En modo gráfico no se puede emplear esta instrucción; debes ponerte antes en modo texto.

La longitud máxima de una línea son 200 caracteres. Este límite lo fija la RAM del sistema. Si quieres ampliarlo deberás emplear la instrucción CLEAR.

PRECAUCIONES

Esta instrucción sólo la puedes emplear con variables de tipo cadena de caracteres. Si la empleas con una variable numérica te dará un error en el tipo de datos *(Type mismatch error)*.

Si la cadena de caracteres leída supera el área de trabajo de dichas cadenas te ocurrirá un error de exceso de caracteres *(Out of string space error)*; el máximo son 200 caracteres.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

LINE INPUT#
INPUT

Parte primera, tema 10: "Programación interactiva".

LINE INPUT

DESCRIPCION

Esta instrucción es similar a INPUT#, pero te permite leer líneas enteras de hasta 200 caracteres de un fichero secuencial que haya en una unidad externa, como es el cassette.

Podrás incluir espacios y comas en las cadenas de caracteres que leas del cassette.

SINTAXIS

LINE INPUT # <número-fichero>,<var-cad>

<número-fichero> ha de ser el número de un fichero abierto (OPEN).

EJEMPLO

```
LINE INPUT # 1,A$
```

PUNTOS A RECORDAR

LINE INPUT # leerá todos los caracteres que encuentre hasta llegar a la marca de fin de línea (<return>).

Es una instrucción bastante útil en los casos en que cada línea del fichero esté subdividida en subcampos.

Puedes leer incluso un programa BASIC grabado en código ASCII línea a línea mediante otro programa que tenga la instrucción LINE INPUT#.

PRECAUCIONES

Sólo puedes utilizar esta instrucción con variables de tipo cadena de caracteres. Si lo haces con variables numéricas tendrás un error en el tipo de datos *(Type mismatch error)*.

Si los caracteres de una línea exceden los del máximo fijado en el área de trabajo de cadenas de caracteres, ocurrirá un error de la capacidad una cadena (el máximo estándar son 200 caracteres).

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INPUT
INPUT#
INPUT$(#)
OPEN
LINE INPUT

Parte segunda, tema 47: "Manejo de ficheros".

LIST

DESCRIPCION

Este comando lista por pantalla el programa BASIC que se encuentra en memoria. El listado puede ser total o parcial.

SINTAXIS

LIST	Lista el programa entero.
LIST <línea>	Lista la línea especificada.
LIST <línea1>-<línea2>	Lista la parte del programa comprendido entre la línea1 y la línea2, inclusive.
LIST <línea>-	Lista desde la línea especificada hasta el final del programa.
LIST -<línea>	Lista desde la primera línea del programa hasta la especificada.
LIST	Lista la última línea a que hizo referencia el ordenador.

EJEMPLOS

LIST 100-200	Lista desde la línea 100 hasta la 200 inclusive.
LIST –100	Lista hasta la línea 100 inclusive.

PUNTOS A RECORDAR

Al conectar el ordenador la tecla F4 realiza la función LIST.

Todas las letras minúsculas de un programa se transformarán a mayúsculas, excepto aquellas que están entre comillas.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

Parte segunda, tema 42: "Gráficos avanzados" (III).

LLIST

DESCRIPCION

Este comando te lista total o parcialmente el programa en BASIC, que se encuentre en memoria, por la impresora.

SINTAXIS

LLIST	Imprime el programa entero.
LLIST <línea>	Imprime la línea especificada.
LLIST <línea1>-<línea2>	Imprime la parte de programa que va desde la línea1 hasta la línea2, inclusive.
LLIST <línea>-	Imprime la parte de programa que va desde la línea hasta el final.
LLIST -<línea>	Imprime desde el principio hasta la línea especificada.
LLIST	Imprime la última línea a que hizo referencia el ordenador.

EJEMPLOS

LLIST 100-200	Imprime desde la línea 100 a la 200, inclusive.
LLIST –100	Imprime hasta la línea 100

PUNTOS A RECORDAR

Todas las letras minúsculas, excepto aquellas que se encuentren entre comillas, serán convertidas en mayúsculas.

PRECAUCIONES

Si la impresora no está conectada no ocurrirá nada.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

LIST

Parte segunda, tema 51: "Periféricos".

LOAD

DESCRIPCION

Esta instrucción carga de la unidad especificada un fichero BASIC, previamente grabado en código ASCII mediante la orden SAVE.

De momento, sólo está implementada la opción de cassette (CAS:). La orden RUN tiene la opción de poder ejecutar el programa una vez cargado especificando R.

SINTAXIS

LOAD"<nombre-unidad>"	Carga el primer fichero BASIC que encuentre, grabado en caracteres ASCII.
LOAD"<nombre-unidad> <nombre-fichero>"	Carga el fichero BASIC especificado.
LOAD"<nombre-unidad> <nombre-fichero>",R	Carga el programa BASIC especificado y lo ejecuta por el momento.

<nombre-unidad> = CAS:,

EJEMPLO

```
LOAD"CAS: JUEGO",R
```

Carga y ejecuta automáticamente el programa JUEGO, grabado en código ASCII.

PUNTOS A RECORDAR

Al ejecutar la instrucción LOAD, el ordenador cerrará todos los ficheros abiertos y cargará un programa BASIC nuevo, que borrará el que había previamente.

<CTRL> <Z> se tratará como una marca de fin de fichero (EOF)

PRECAUCIONES

Para cargar bytes emplea la instrucción BLOAD y no LOAD.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SAVE
CLOAD
CSAVE
BLOAD
BSAVE

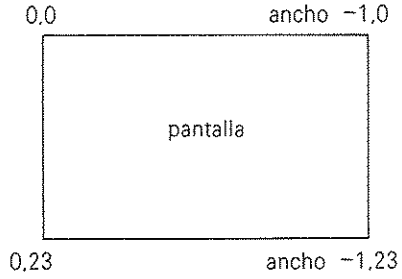
Parte segunda, tema 39: "Grabación y carga con el cassette".

LOCATE

DESCRIPCION

La instrucción LOCATE sitúa el cursor en la posición especificada; también puede hacer que el cursor se visualice o no. Se suele usar en combinación con la instrucción PRINT, para poder mostrar parte de un texto en una determinada posición de la pantalla.

Puedes mover el cursor a cualquier punto de la pantalla que no exceda de los límites del modo texto ni de la instrucción WIDTH.



<--ancho asignado-->
mediante WIDTH o el ancho
estándar (que depende del modo texto)

La anchura estándar para ambos modos de texto es:

MODO 0: Ancho = 37 (máximo = 40)
MODO 1: Ancho = 29 (máximo = 32)

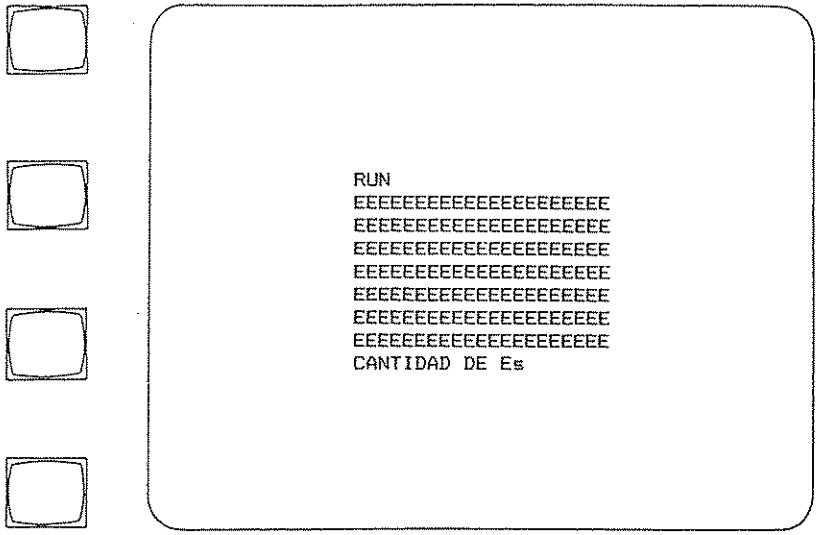
SINTAXIS

LOCATE <coordenada x>, <coordenada y>	Mueve el cursor a las coordenadas especificadas.
LOCATE <coordenada x>, <coordenada y>,0	Mueve el cursor a la posición especificada, pero no lo visualiza.
LOCATE <coordenada x>, <coordenada y>,1	Mueve el cursor a la posición especificada, visualizándolo.

<coordenada x> y <coordenada y> pueden ser variables, constantes o expresiones, pero con resultado dentro de los límites de la pantalla. Todos los números reales se truncan a enteros.

EJEMPLO

```
10 FOR Y=0 TO 6
20 FOR X=0 TO 20
30 LOCATE X,Y
40 PRINT"E"
50 NEXT:NEXT
60 PRINT "CANTIDAD DE Es"
```



PUNTOS A RECORDAR

LOCATE no funcionará como tal en los modos gráficos.

Se puede emplear para situar un mensaje determinado de entrada (INPUT) en una posición concreta.

Existe una función parecida, TAB, que se usa de la forma PRINT TAB, pero que es más restringida; no te permite mover el cursor en la dirección del eje Y; o sea, que LOCATE te será bastante más útil.

PRECAUCIONES

Si las coordenadas desbordan los límites de la pantalla se producirá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT
INPUT
TAB

Parte primera, tema 9: "Algo más sobre las salidas (PRINT) y la pantalla".

LOG

DESCRIPCION

Esta función devuelve como resultado el logaritmo neperiano de un número en doble precisión.

SINTAXIS

LOG(<num-const>)
LOG(<num-var>)
LOG(<num-exp>)

EJEMPLO

```
PRINT LOG(10)
2.302585092994
```

PUNTOS A RECORDAR

Esta función calcula el logaritmo neperiano, cuya base es el número *e*; por tanto, es completamente diferente al logaritmo en base 10.

PRECAUCIONES

Si el argumento de la función fuese negativo se producirá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

EXP

Parte primera, tema 19: "Funciones matemáticas".

LPOS POSición en Línea

DESCRIPCION

LPOS nos proporciona la posición en la que se encuentre la cabeza de escritura en el *buffer* de escritura.

Esta función necesita de un argumento X sin significado alguno.

LPOS no representa, necesariamente, la posición física de dicha cabeza.

SINTAXIS

LPOS(X)

EJEMPLOS

```
PRINT LPOS(X)
```

o

```
A=LPOS(X)
```

PUNTOS A RECORDAR

A esta función no puede asignársele ningún valor, ya que se trata de una variable del sistema.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 51: "Periféricos".

LPRINT

DESCRIPCION

Es la instrucción para escribir por impresora; en líneas generales tiene el mismo funcionamiento que la instrucción PRINT.

SINTAXIS

LPRINT

EJEMPLO

LPRINT "Esto es una prueba";123

que hará que salga por impresora: "Esto es una prueba 123".

PUNTOS A RECORDAR

No olvides que para usar esta instrucción ha de estar conectada la impresora.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT
LPRINT USING

Parte segunda, tema 51: "Periféricos".

LPRINT USING

DESCRIPCION

LPRINT USING es la misma instrucción que PRINT USING, excepto que la salida se lleva a cabo por impresora. (Véase PRINT USING.)

SINTAXIS

(Véase PRINT USING.)

EJEMPLO

LPRINT USING "MI ORDENADOR @ ES LO QUE MAS QUIERO";"MSX"

imprimirá:

MI ORDENADOR MSX ES LO QUE MAS QUIERO.

PUNTOS A RECORDAR

Cuando emplees la instrucción no te olvides de tener conectada la impresora.

PRECAUCIONES

(Véase PRINT USING.)

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

LPRINT
PRINT USING

Parte segunda, tema 36: "PRINT USING".
Parte segunda, tema 51: "Periféricos".

MAXFILES Número máximo de ficheros *(MAXimum number of FILES)*

DESCRIPCION

Fija el número máximo de ficheros sobre los que se puede estar trabajando en un momento dado. Este número de ficheros ha de estar comprendido entre 0 y 15. Si a la variable MAXFILES no le asignas un valor, el ordenador asume por defecto el valor 1.

Si esta variable tiene el valor 0, entonces sólo podrás grabar (SAVE) y cargar en memoria (LOAD).

SINTAXIS

MAXFILES = <num-const>
MAXFILES = <num-var>
MAXFILES = <num-exp>

Rango de 0 a 15.

EJEMPLOS

```
1000 MAXFILES=5
```

PUNTOS A RECORDAR

La labor de MAXFILES es la de aumentar el bloque de control de ficheros. Estos bloques de control se emplean cuando estás trabajando con ficheros que se encuentran almacenados en periféricos.

PRECAUCIONES

El error más corriente será el de olvidarte de la S de MAXFILES.

Si el argumento no está comprendido entre 0 y 15, se producirá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OPEN
CLOSE

Parte segunda, tema 47: "Manejo de ficheros".

MERGE

DESCRIPCION

Esta instrucción te permite fundir dos programas en uno.

Supongamos que tenemos dos programas BASIC, programa 1 y programa 2, y los quieres unir de modo que el programa 2 quede a continuación del programa 1. Supongamos también que ambos están grabados en cassette.

Primero: carga el programa 2 en memoria y cambia todos los números de línea de modo que sean todos mayores que los del programa 1. Ahora graba en código ASCII el programa 2 en cassette, mediante el comando SAVE.

Carga el programa 1, teclea MERGE "programa" y pulsa la tecla *play* del cassette. El ordenador solapará el programa 2 al final del programa 1.

Si coincidieran alguno de los números de línea, el programa resultante contendrá la línea del programa 2.

SINTAXIS

MERGE"<nombre-unidad>"
MERGE"<nombre-unidad><nombre-fichero>"
<nombre-unidad>=CAS: para el cassette.

EJEMPLOS

Cómo unir dos programas, el 1 y el 2.

PROGRAMA 1

```
10 PRINT "HOLA"
20 PRINT "BIENVENIDO"
30 PRINT "AL"
```

PROGRAMA 2

```
10 FOR I=1 TO 10
20 PRINT "MSX"
30 NEXT I
```

Carga en memoria el programa 2(CLOAD) y renuméralo desde la línea 50(RENUM 50). Si lo listas tendrás:

```
40 FOR I=1 TO 10
50 PRINT "MSX"
60 NEXT I
```

Graba en formato ASCII el programa 2 mediante: SAVE"CAS:PROG2". Ahora carga el programa 1 con CLOAD.

MERGE"CAS:PROG2" añadirá el programa 2 a continuación del programa 1. Si listas el programa almacenado en memoria tendrás:

```
10 PRINT "HOLA"
20 PRINT "BIENVENIDO"
30 PRINT "AL"
40 FOR I=1 TO 10
50 PRINT "MSX"
60 NEXT I
```

PUNTOS A RECORDAR

Si en ambos programas existiese una línea con el mismo número, el programa resultante de la unión tendrá la línea del programa cargado con MERGE, el programa (2).

Si en esta instrucción no especificamos el nombre del fichero, el ordenador solapará al programa cargado en memoria el primer fichero ASCII que encuentre.

<CTRL> <Z> indica al ordenador el final del fichero (EOF).

PRECAUCIONES

Si en la instrucción MERGE tecleas mal el nombre del programa, el ordenador se lo saltará, y no lo unirá al residente en memoria.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SAVE

Parte segunda, tema 39: "Grabación y carga con el cassette".

MID$

DESCRIPCION

Esta función devuelve una parte de una cadena de caracteres.

SINTAXIS

MID$(<var-cad>,X,Y) Te devuelve la cadena de longitud Y que comienza en la posición X de la cadena asignada como argumento.

MID$(<var-cad>,X) Te devuelve la parte de la cadena que se especifica desde la posición X.

EJEMPLO

MID$ se suele emplear para separar palabras de una misma sentencia. Aquí tienes un ejemplo:

```
10 A$="MARTIN LUTERO KING"
20 B1=INSTR(A$," ")
30 B2=INSTR(B1+1,A$," ")
40 PRINT LEFT$(A$,B1-1)
50 PRINT MID$(A$,B1+1,B2-B1-1)
60 PRINT MID$(A$,B2+1)
```

```
RUN
MARTIN
LUTERO
KING
```

PUNTOS A RECORDAR

X e Y son dos valores enteros comprendidos entre 1 y 255.

Si el valor de X es mayor que la longitud de la cadena asignada, entonces esta función te dará como resultado una cadena vacía ("").

PRECAUCIONES

Esta función sólo trabaja con cadenas de caracteres, es decir, que ya sabes cómo evitar errores con los tipos de datos.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

RIGHT$
LEFT$
LEN
INSTR

Parte primera, tema 14: "Las cadenas de caracteres".

MOD Resto *(MODulus)*

DESCRIPCION

Esta instrucción devuelve el resto de la división de dos números enteros.

```
10 MOD 3 = 1
```

ya que

10/3 es 3 con un resto 1.

SINTAXIS

<var-num> = <número> MOD <número>

EJEMPLO

```
10 PRINT 15 MOD 5
```

el resultado será cero.

PUNTOS A RECORDAR

Los operandos son truncados a enteros antes de que sean operados.

PRECAUCIONES

Si mezclas esta operación con cadenas de caracteres se producirá un error en el tipo de datos *(Type mismatch error)*.

En caso de que alguno de los argumentos se salga del rango especificado (−32768, 32767) se producirá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Y(DIV)

Parte segunda, tema 32: "Operadores y expresiones".

MOTOR

DESCRIPCION

Esta instrucción permite el manejo del cassette mediante control remoto, estando previamente conectados el ordenador y el cassette mediante la conexión de control remoto.

La instrucción MOTOR se refiere al motor del cassette.

SINTAXIS

MOTOR	Permite o inhibe el trabajo independiente del cassette, según esté éste inhibido o permitido, respectivamente.
MOTOR ON	Permite el trabajo independiente del cassette.
MOTOR OFF	Inhibe el trabajo independiente del cassette.

PUNTOS A RECORDAR

Esta instrucción te será útil en el caso de que tu cassette posea la entrada REM.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

(Véase la sección correspondiente al cassette.)

NEW

DESCRIPCION

Esta instrucción borra el programa almacenado en memoria, inicializa todas las variables y el puntero de pila, etc. Una vez que hayas ejecutado esta instrucción no tendrás acceso al programa anterior. Prepara al ordenador para un nuevo programa.

SINTAXIS

NEW

EJEMPLO

NEW

PUNTOS A RECORDAR

Esta instrucción no borra la pantalla.

Ejecuta la instrucción NEW antes de cargar un programa nuevo, ya que se podrían mezclar las variables del programa anterior con las del nuevo.

PRECAUCIONES

Es fatal que ejecutes NEW por error.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 3: "Escritura de un programa".

NEXT

DESCRIPCION

NEXT es parte de la instrucción FOR, e indica al ordenador que vuelve a su FOR correspondiente o salga del bucle si fuese la última iteración.

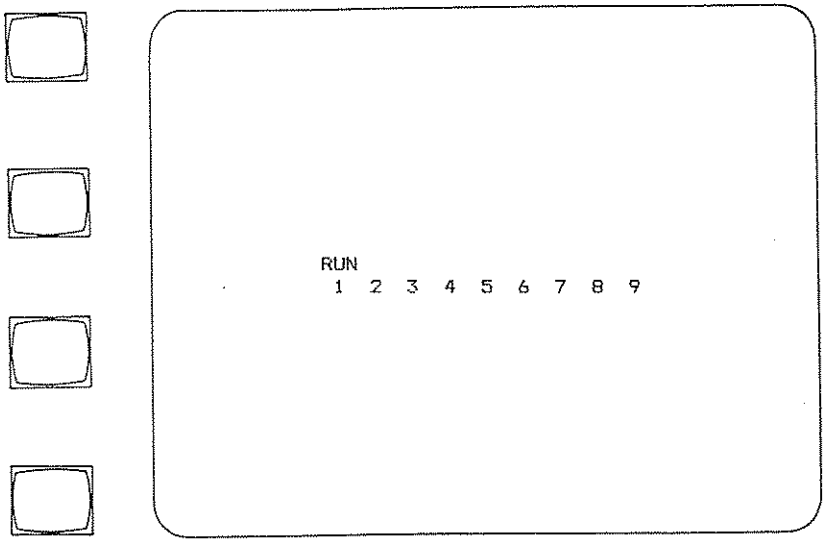
Si quieres informarte más extensamente sobre el bucle FOR/NEXT consulta el capítulo correspondiente a FOR.

SINTAXIS

NEXT
NEXT <variable>
NEXT <variable>,<variable>... lista de variables.

EJEMPLO

```
10 FOR I=1 TO 9
20 PRINT I;
30 NEXT I
```



PUNTOS A RECORDAR

La variable después de NEXT puede omitirse.

PRECAUCIONES

El siguiente ejemplo te muestra un error corriente con la instrucción NEXT; NEXT I y NEXT J se encuentran en orden inverso.

```
10 FOR I=1 TO 5
20 FOR J=1 TO 10
30 PRINT I,J
40 NEXT I
50 NEXT J
```

LINEA 40 ERROR DE NEXT SIN FOR CORRESPONDIENTE.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

FOR
TO
STEP

Parte primera, tema 5: "Cómo emplear los bucles".

NOT

DESCRIPCION

Se emplea en las condiciones de la instrucción IF/THEN/ELSE y también en operaciones lógicas. Devuelve el valor contrario de su argumento, es decir, NOT (VERDADERO) es FALSO.

Tabla de verdad del operador NOT.

NOT	X	NOT X
	0	1
	1	0

SINTAXIS

<var-num> = NOT(<número>)
IF NOT(<condición>) THEN ...

EJEMPLOS

```
A=0
PRINT NOT(A)
```

que dará como resultado −1. A = &B0000000000000000 en binario. NOT(A) = &B1111111111111111 que es −1 en decimal.

```
IF NOT(A=100 AND B>900) THEN GOTO 10000
```

PUNTOS A RECORDAR

El argumento ha de ser un entero comprendido entre −32768 y 32767.
Si el argumento es un número real será truncado al entero más próximo.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

IF
THEN
ELSE
AND
OR
XOR
IMP
EQV

Parte segunda, tema 34: "Algebra de Boole" (I).
Parte segunda, tema 35: "Algebra de Boole (II): la instrucción IF/THEN/ELSE".

OCT$

DESCRIPCION

La función OCT$ devuelve una cadena de caracteres con el valor octal (base 8) del argumento decimal pasado.

El sistema de numeración octal es aquel cuya base es 8. Este sistema emplea solamente los números 0, 1, 2, 3, 4, 5, 6, 7, y cada posición del número octal representa una determinada potencia de 8.

El argumento ha de ser una expresión numérica comprendida entre −32768 y 65535.

SINTAXIS

<var-cad> = OCT$(<número>)

EJEMPLO

```
10 PRINT OCT$(8)
20 PRINT OCT$(&HF)
30 PRINT OCT$(8^4)
40 PRINT OCT$(8^4-1)
```

```
10
17
10000
7777
```

PUNTOS A RECORDAR

Si el argumento fuese negativo, el resultado se daría en complemento a dos; es decir, OCT$(−X) = OCT$(65535 − X).

PRECAUCIONES

Si el argumento se encuentra fuera del rango permitido, se producirá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

BIN$
HEX$

Parte segunda, tema 33: "Presentación de los sistemas de numeración del MSX".

ON <expresión> GOSUB

DESCRIPCION

Esta instrucción ofrece la posibilidad de salto a diferentes y variadas subrutinas. La subrutina a la que se produce el salto depende de la expresión. Si ésta vale 1 el ordenador saltará a la primera subrutina; si valiese 2 iría a la segunda, y así sucesivamente.

El resultado de la expresión ha de estar comprendido entre 1 y el número de subrutinas apuntadas en la instrucción.

Por ejemplo:

```
ON X GOSUB 100,250,670,800
```

Si X vale 1 saltará a la subrutina que comienza en la línea 100.
Si X vale 2 saltará a la subrutina que comienza en la línea 250.
Si X vale 3 saltará a la subrutina que comienza en la línea 670.
Si X vale 4 saltará a la subrutina que comienza en la línea 800.

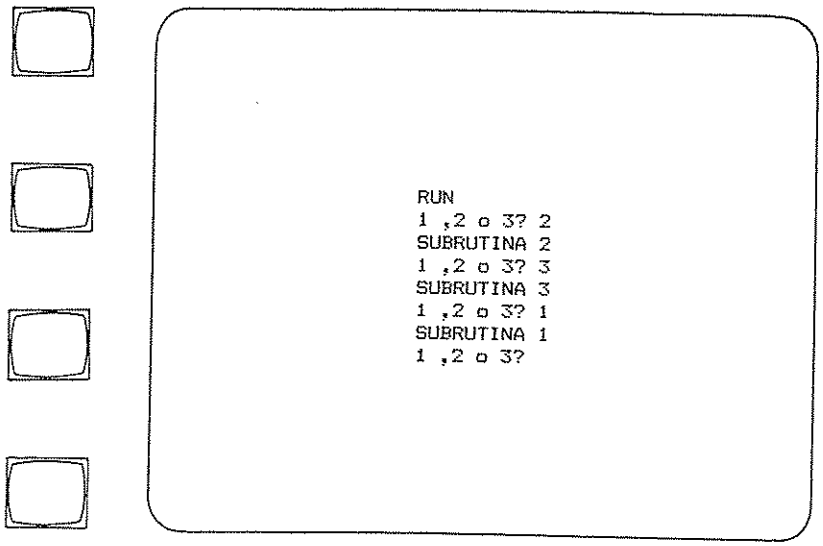
Si X tomase el valor 0 o un valor superior el número de subrutinas apuntadas (en el caso anterior 4), entonces el ordenador continuaría ejecutando la siguiente instrucción a ON <expresión> GOSUB...

SINTAXIS

ON <expresión> GOSUB <lista de números de línea>

EJEMPLO

```
10 INPUT "1 ,2 o 3";I
20 ON I GOSUB 40,60,80
30 GOTO 10
40 PRINT "SUBRUTINA 1"
50 RETURN
60 PRINT "SUBRUTINA 2"
70 RETURN
80 PRINT "SUBRUTINA 3"
90 RETURN
```



```
RUN
1 ,2 o 3? 2
SUBRUTINA 2
1 ,2 o 3? 3
SUBRUTINA 3
1 ,2 o 3? 1
SUBRUTINA 1
1 ,2 o 3?
```

PUNTOS A RECORDAR

Si el resultado de la expresión es un número real, entonces se trunca su parte decimal y se ejecuta la subrutina correspondiente; es decir, si <expresión> = 1.22, entonces se hace <expresión> = 1 y se ejecutará la primera subrutina.

Si <expresión> = 0, o es mayor que el número de subrutinas apuntadas en la lista, y es menor que 255, y entonces el BASIC continuará con la siguiente instrucción.

PRECAUCIONES

Si el resultado de la <expresión> fuese negativo o mayor que 255, entonces se produciría un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON ... GOTO

Parte primera, tema 18: "Programas de bifurcación".

ON <expresión> GOTO

DESCRIPCION

La instrucción ON <expresión> GOTO <línea1>, <línea2>, <línea3>, ..., ofrece la posibilidad de realizar saltos condicionales múltiples. El salto elegido depende de la <expresión>: si la <expresión> vale 1, entonces se continuará con la primera línea apuntada; si vale 2 se continuará con la segunda, y así sucesivamente.

El resultado de la expresión ha de estar comprendido entre 1 y el número de líneas apuntadas en la instrucción.

Por ejemplo:

ON X GOTO 100, 250, 670, 800

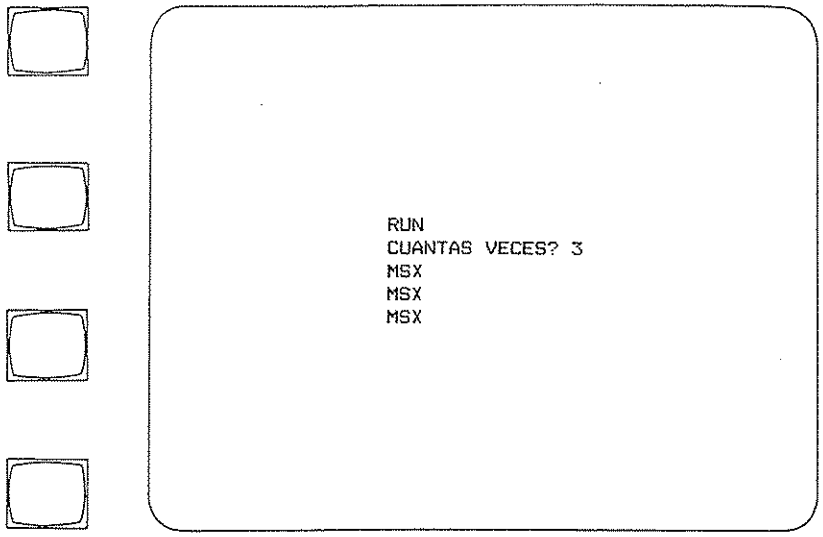
Si X vale 1 se saltará a la línea 100.
Si X vale 2 se saltará a la línea 250.
Si X vale 3 se saltará a la línea 670.
Si X vale 4 se saltará a la línea 800.
Si X valiese 0 o más que el número de líneas apuntadas, en este caso 4, entonces se continuaría con la siguiente instrucción.

SINTAXIS

ON <expresión> GOTO <lista de números de línea>

EJEMPLO

```
10 INPUT "CUANTAS VECES";N
20 ON N GOTO 60,50,40
30 GOTO 10
40 PRINT "MSX"
50 PRINT "MSX"
60 PRINT "MSX"
```



```
RUN
CUANTAS VECES? 3
MSX
MSX
MSX
```

PUNTOS A RECORDAR

Si el resultado de <expresión> fuese un número real, se le truncaría su parte decimal; es decir, si <expresión> = 1.78, entonces se haría <expresión> = 1 y se saltaría a la primera línea de la lista.

Si <expresión> = 0 o <expresión> mayor que el número de líneas apuntadas y menor que 255, entonces el BASIC continuará con la siguiente instrucción a ON <expresión> GOTO ...

PRECAUCIONES

Si el resultado de <expresión> fuese negativo o mayor que 255 se produciría un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON ... GOSUB

Parte primera, tema 18: "Programas de bifurcación".

ON ERROR GOTO

DESCRIPCION

ON ERROR GOTO activa la detección de errores e indica a qué línea se ha de saltar en caso de que se produjese un error. Si se produjera durante la ejecución de un programa, éste saltará a la línea especificada por esta instrucción.

Para desactivar la detección de errores has de ejecutar ON ERROR GOTO 0. Si la ejecutas dentro de la rutina de detección de errores el BASIC se detendrá y te imprimirá el error que haya detectado.

Mediante la instrucción ERROR puedes crear tus propios errores. El BASIC del MSX tiene unos 36 errores con códigos entre 1 y 60; es decir, que los códigos de error entre 60 y 255 están a tu disposición.

Para programar tus propios errores ejecuta primeramente la instrucción ON ERROR GOTO <línea>; entonces, si en el programa se cumple cierta condición que desees tratar como un error, ejecuta:

IF <condición> THEN ERROR <código de error>

La rutina de detección de errores se pondrá en marcha y saltará a la subrutina de tratamiento de errores. En la subrutina de tratamiento de errores deberá haber como mínimo una instrucción como ésta:

IF ERR=<código de error> THEN PRINT "mensaje de error"

Esta subrutina puede terminar la ejecución del programa, o bien continuar con él en una determinada línea (RESUME).

SINTAXIS

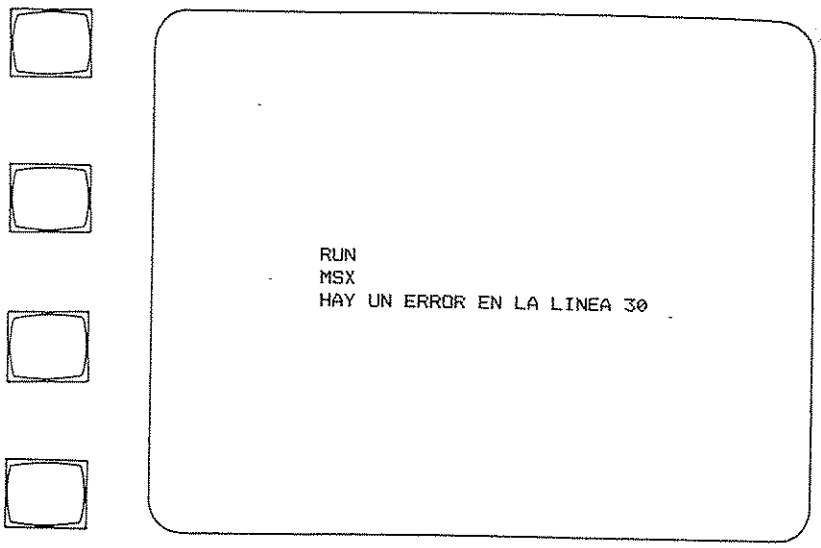
ON ERROR GOTO <línea>

EJEMPLO

Aquí tienes una sencilla detección de errores:

```
10   ON ERROR GOTO 1000
20   PRINT "MSX"
30   PRONT "ERROR"
40   END
1000 PRINT "HAY UN ERROR EN LA LINEA";ERL
1010 END
```

LINEA 30 TIENE UN ERROR.



PUNTOS A RECORDAR

El MSX no detectará los errores hasta que esté ejecutando la subrutina de detección de errores. Si éstos son detectados dentro de la subrutina de detección, entonces el BASIC se detendrá y dará un mensaje de error.

No todos los errores pueden ser corregidos desde su subrutina de tratamiento; por tanto, te aconsejamos que siempre termines con la instrucción ON ERROR GOTO 0, que detendrá el BASIC y te mostrará un mensaje de error.

Una vez que hayas activado la subrutina de detección de errores el ordenador saltará a la subrutina de tratamiento cuando haya alguno, incluso en modo directo. Si estás trabajando con tus propios errores te recomendamos que les asignes códigos de error, a partir de 255 y de forma descendente, es decir, 255, 254, 253, ...

Si el ordenador detecta un error que no tenga mensaje de error se te mostrará el mensaje "*Unprintable error*" (error sin mensaje).

La instrucción ON ERROR desactiva todas las posibles subrutinas de interrupción como ON INTERVAL u ON STRIG.

PRECAUCIONES

Si no pones el número de línea tendrás un error de número de línea no definido *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ERL
ERR
ERROR
RESUME

Parte segunda, tema 38: "Tratamiento de errores".

ON INTERVAL GOSUB

DESCRIPCION

Esta instrucción fija la subrutina de tratamiento de interrupciones temporales y el intervalo de tiempo con que se producen dichas interrupciones. Cuando estén activadas las interrupciones de tiempo (con INTERVAL ON), el ordenador —cada cierto intervalo de tiempo— saltará a la subrutina de tratamiento de dichas interrupciones. Esta es una de las instrucciones de interrupción en que está especializado el MSX. (Para una explicación más extensa consulta la sección de manejo de sucesos e interrupciones.)

La unidad de tiempo del intervalo es 1/50 de segundo, y según el valor que tenga asignado podrá tener toda la longitud que se desee.

Una vez que se produce la interrupción, se ejecuta automáticamente un INTERVAL STOP. Esto evita que, aparte de la interrupción producida, se traten otras mientras dura aquélla; sin embargo, recuerda si se ha producido una interrupción, y pasará a ejecutar otra vez su subrutina de tramiento, excepto si durante esta subrutina se ha ejecutado la instrucción INTERVAL OFF. Después de ejecutar la subrutina de tratamiento de interrupciones el ordenador ejecutará automáticamente una orden INTERVAL ON.

SINTAXIS

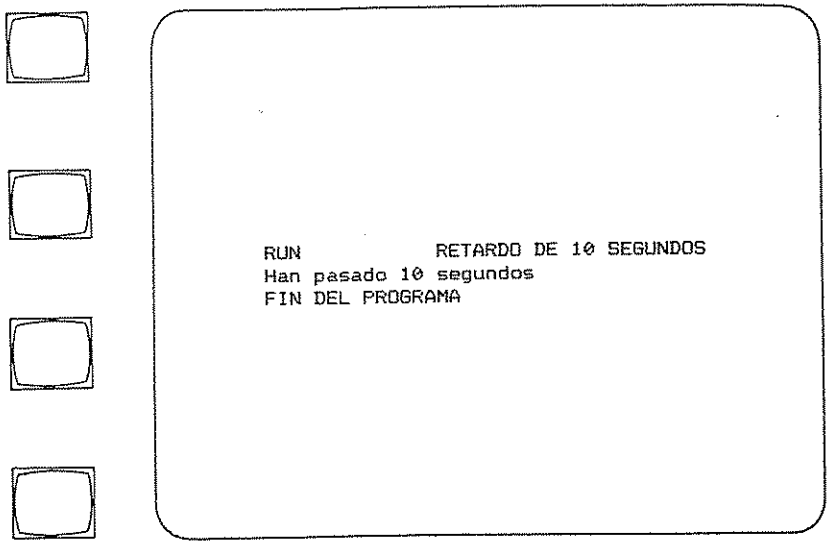
ON INTERVAL= <intervalo de tiempo en 1/50 de segundo> GOSUB <línea>

EJEMPLOS

```
10 ON INTERVAL=500 GOSUB 40
20 INTERVAL ON
30 GOTO 30
35 REM Subrutina de intervalo
40 PRINT "Han pasado 10 segundos"
50 PRINT "FIN DEL PROGRAMA"
60 END
```

LINEA 10 FIJA EL INTERVALO A 500 UNIDADES (10 SEGUNDOS) Y LA SUBRUTINA DE TRATAMIENTO EN LA LINEA 40.
LINEA 20 ACTIVA LA INTERRUPCION.

Ejecuta el programa (RUN) y obtendrás:



```
RUN             RETARDO DE 10 SEGUNDOS
Han pasado 10 segundos
FIN DEL PROGRAMA
```

PUNTOS A RECORDAR

Las interrupciones de tiempo estarán desactivadas siempre y cuando no se esté ejecutando un programa en BASIC; esto es, no habrá interrupciones temporales en modo directo.

También estarán desactivadas mientras que se encuentren activadas las subrutinas de detección de errores.

PRECAUCIONES

No olvides activar las interrupciones con INTERVAL ON.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

INTERVAL/ON/OFF/STOP

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

ON KEY GOSUB

DESCRIPCION

ON KEY GOSUB fija los números de línea de comienzo de las subrutinas de tratamiento de las interrupciones de las teclas de función. KEY (<número>) ON activa la detección de interrupciones de las teclas de función. Cuando éstas se producen el ordenador salta a la línea correspondiente especificada por ON KEY GOSUB.

Para cada tecla de función has de indicar la línea de su correspondiente subrutina. Si no hay una subrutina para una determinada tecla, sáltate el número de línea y pon una coma.

Una vez que se produce la interrupción de una tecla de función se ejecuta automáticamente la instrucción KEY (<número>) STOP. Con ello se evita que se trate otra interrupción de esa misma tecla durante el tratamiento de la interrupción actual. Sin embargo, el ordenador memoriza si se ha producido una interrupción de la tecla tratada y, una vez ejecutada la subrutina de la interrupción actual, vuelve a ejecutar dicha subrutina para la siguiente interrupción (siempre que sea la misma tecla de función), excepto si en dicha subrutina se ha desactivado la detección de interrupciones de esa tecla ejecutando KEY (<número>) OFF. Después de ejecutar la subrutina el ordenador ejecutará automáticamente la instrucción KEY ON, activando de nuevo las interrupciones.

SINTAXIS

ON KEY GOSUB <lista de números de línea>

EJEMPLOS

Si pulsas la tecla de función 1, el ordenador emitirá dos sonidos *(beeps)*; si pulsas la tecla de función 2, sólo emitirá uno.

```
10 ON KEY GOSUB 50,60
20 KEY(1) ON
30 KEY(2) ON
40 GOTO 40
50 BEEP
60 BEEP
70 RETURN
```

LINEA 10 FIJA LAS SUBRUTINAS DE F1 Y F2 EN LAS LINEAS 50 Y 60, RESPECTIVAMENTE.
LINEA 20 ACTIVA LA TECLA DE FUNCION F1.
LINEA 30 ACTIVA LA TECLA DE FUNCION F2.
LINEA 40 BUCLE INFINITO EN ESPERA DE UNA INTERRUPCION.
LINEA 50 SONIDO (BEEP) DE F1.

LINEA 60 SONIDO (BEEP) DE AMBAS FUNCIONES.
LINEA 70 RETORNO AL PUNTO DE PARTIDA, EN ESTE CASO LA LINEA 40.

PUNTOS A RECORDAR

Las interrupciones de teclas de función se desactivan automáticamente durante la ejecución de subrutinas de detección de errores.

PRECAUCIONES

Si al poner los números de las líneas en la lista te equivocas, se producirá un error de línea no definida *(Undefined line number)*.

No te olvides de KEY ON.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

KEY/ON/OFF/STOP

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

ON SPRITE GOSUB

DESCRIPCION

Esta instrucción fija la subrutina de interrupción por choque de *sprites*. La instrucción SPRITE ON permite la detección y salto a la subrutina de tratamiento cuando chocan dos *sprites*.

Cuando ocurre la interrupción se ejecuta automáticamente la instrucción SPRITE STOP. Esto evita que se traten otras interrupciones por colisión de *sprites* durante la ejecución de dicha subrutina. Sin embargo, el ordenador memoriza si se ha producido otro choque y lo trata una vez terminada la ejecución de la anterior subrutina, a menos que en ésta se haya ejecutado una instrucción SPRITE OFF. Después de la ejecución de la subrutina de tratamiento de choques se ejecutará automáticamente SPRITE ON para activar de nuevo estas interrupciones.

SINTAXIS

ON SPRITE GOSUB <línea>

EJEMPLO

Con este ejemplo verás dos *sprites* cuadrados, uno amarillo y otro blanco, aproximándose desde cada lado de la pantalla. Cuando choquen, ON SPRITE GOSUB hará que el BASIC salte a la subrutina de interrupción. SPRITE OFF evita que se vuelva a detectar cualquier otra colisión entre *sprites*. (Prueba el programa sin SPRITE OFF y observa lo que ocurre.)

```
10  ON SPRITE GOSUB 110
20  SCREEN 2,0
30  SPRITE$(0)=STRING$(8,CHR$(255))
40  SPRITE$(1)=STRING$(8,CHR$(255))
50  SPRITE ON
60  FOR I=10 TO 240
70  PUT SPRITE 0,(I,100),11,0
80  PUT SPRITE 1,(250-I,100),15,1
90  NEXT I
100 END
105 REM Rutina de choque de sprites
110 SPRITE OFF
120 BEEP
130 RETURN
```

LINEA 10 FIJA LA SUBRUTINA DE CHOQUE DE SPRITES EN LA LINEA 110.
LINEA 20 CAMBIA LA PANTALLA A MODO GRAFICO.
LINEA 30 EL SPRITE 0 ES UN CUADRADO.
LINEA 40 EL SPRITE 1 ES OTRO CUADRADO.

LINEA 50 ACTIVA LA DETECCION DE CHOQUES DE SPRITES.
LINEA 60 BUCLE.
LINEA 70 SPRITE AMARILLO MOVIENDOSE DESDE LA IZQUIERDA.
LINEA 80 SPRITE BLANCO MOVIENDOSE DESDE LA DERECHA.
LINEA 90 SIGUIENTE BUCLE.
LINEA 100 FIN.
LINEA 110 DESACTIVA LA DETECCION DE MAS CHOQUES.
LINEA 120 SONIDO INDICADOR DE CHOQUE.
LINEA 130 RETORNO AL PUNTO DE DONDE VINO.

PUNTOS A RECORDAR

Las interrupciones por choque de *sprites* se encuentran desactivadas mientras el programa no se ejecute, y también durante la ejecución de subrutinas de detección de errores.

PRECAUCIONES

Si el número de línea donde comienza la subrutina de tratamiento del choque está equivocado, se producirá un error de línea no definida *(Undefined line number)*.

No te olvides de ejecutar SPRITE ON.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SPRITE ON/OFF/STOP
SPRITE$
PUT SPRITE

Parte segunda, tema 43: "Gráficos avanzados (IV): los *sprites*".

ON STOP GOSUB

DESCRIPCION

ON STOP GOSUB es una instrucción de manejo de interrupciones, que evita que el usuario detenga la ejecución de un programa mediante <CTRL><STOP>.

El único modo de detener un programa con esta instrucción es el de inicializar (RESET) la máquina. Por tanto, no olvides grabar tu programa a prueba de paradas antes de ejecutarlo (RUN).

STOP ON/OFF/STOP permite o inhibe la detección de la tecla <STOP>. Si ejecutamos STOP ON, el ordenador analiza si se ha pulsado <CTRL><STOP> antes de ejecutar cada instrucción; asi así fuese, el BASIC saltará a la subrutina apuntada por la instrucción ON STOP GOSUB ejecutada anteriormente.

Cuando se produce la interrupción se ejecuta la instrucción STOP STOP automáticamente. Esto evita que durante la ejecución de la subrutina de tratamiento de interrupción se vuelva a tratar otra interrupción; no obstante, el ordenador memoriza si esa interrupción se ha producido, volviendo a ejecutar la subrutina de tratamiento una vez que haya terminado, excepto en el caso que en dicha subrutina tuviésemos una instrucción STOP OFF. Después de salir de la subrutina de tratamiento se ejecutará la instrucción STOP ON, que permitirá la detección de más interrupciones de este tipo.

SINTAXIS

ON STOP GOSUB <línea>

EJEMPLO

Este programa te muestra cómo la instrucción ON STOP GOSUB evita que puedas detener su ejecución. Si pulsas <CTRL><STOP> te contestará que está desactivado y continuará la ejecución del programa. Sin embargo, hay un método de pararlo: pulsando "s"; si no, se imprimirá "MSX" indefinidamente.

```
10  ON STOP GOSUB 100
20  STOP ON
30  IF INKEY$="s" THEN END
40  PRINT "MSX"
50  GOTO 30
90  REM Rutina de <CTRL><STOP>
100 BEEP
110 PRINT "<CTRL><STOP> desactivado"
120 RETURN
```

LINEA 10 FIJA LA SUBRUTINA DEL STOP EN LA LINEA 100.
LINEA 20 ACTIVA LA DETECCION DEL STOP.

LINEA 30 SI PULSAS S EL PROGRAMA TERMINA.
LINEA 40 IMPRIME EN PANTALLA MSX.
LINEA 50 VUELVE A LA LINEA 30.
LINEA 100 SONIDO INDICANDO PROHIBICION (BEEP).
LINEA 110 MENSAJE.
LINEA 120 VUELVE A DONDE SE PRODUJO LA INTERRUPCION.

PUNTOS A RECORDAR

La instrucción ON STOP no prevé la detección del programa con la tecla <STOP>. Sólo previene la detección del programa mediante <CTRL> <STOP>. Prueba a pulsar la tecla <STOP> en el ejemplo anterior: observarás que el programa se detiene y aparece el cursor. Si vuelves a pulsar <STOP> el programa continuará ejecutándose donde estaba.

Para activar la detección de una interrupción de parada debes ejecutar la instrucción STOP ON.

Estas interrupciones se desactivan cuando el programa no se está ejecutando; también dentro de las subrutinas de detección de errores.

PRECAUCIONES

Si apuntas a una línea de comienzo de subrutina que no es correcta tendrás un error de número de línea no definida *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

STOP ON/OFF/STOP

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

ON STRIG GOSUB

DESCRIPCION

Esta instrucción fija las subrutinas de tratamiento de interrupciones del *joystick* y la barra espaciadora. STRIG(<n>) ON activa la detección de estas señales. Se usa conjuntamente con ON STRIG GOSUB.

Hay cinco señales:

0 = barra espaciadora.
1 = señal 1 del *joystick* 1.
2 = señal 1 del *joystick* 2.
3 = señal 2 del *joystick* 1.
4 = señal 2 del *joystick* 2.

En esta instrucción debes enumerar todas las líneas donde comienzan las subrutinas de cada señal, excepto si empleas sólo la barra espaciadora (véase la parte de sintaxis). Si determinada señal no tiene subrutina, omítela y pon en su lugar una coma.

Una vez que se detecta la interrupción se ejecuta automáticamente la instrucción STRIG(<n>) STOP. Con ello se evita el tratamiento de interrupciones de este tipo mientras se está tratando la actual. Sin embargo, el ordenador memoriza si se ha producido una de estas interrupciones y pasa a tratarla una vez haya acabado con la actual, excepto si en la subrutina hay una instrucción STRIG(<n>) OFF. Una vez que se ha ejecutado la subrutina se ejecutará la instrucción STROG(<n>) ON con el objetivo de permitir la detección de interrupciones de ese disparador de señal.

SINTAXIS

ON STRIG GOSUB <lista de números de línea>
ON STRIG GOSUB <línea> (sólo en el caso en que la única señal sea la barra espaciadora).

EJEMPLO

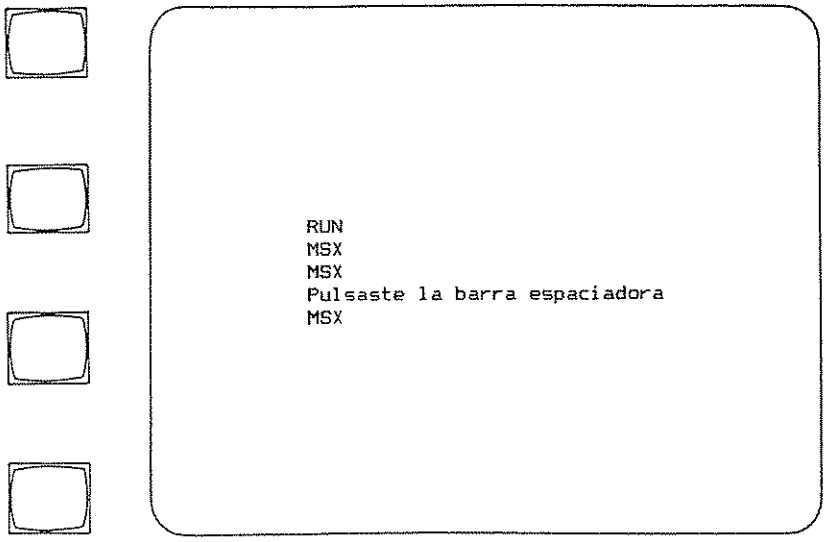
Este pequeño ejemplo te enseña cómo detectar las señales de la barra espaciadora como disparador de señales. Cuando pulses la barra espaciadora el programa saltará a la subrutina correspondiente.

De otro modo se imprimirá indefinidamente "MSX", hasta que presiones la tecla "s".

```
10  ON STRIG GOSUB 100
20  STRIG(0) ON
30  IF INKEY$="s" THEN END
40  PRINT "MSX"
50  GOTO 30
```

```
90  REM Rutina de la barra espaciadora
100 BEEP
110 PRINT "Pulsaste la barra espaciadora"
120 RETURN
```

LINEA 10 FIJA LA SUBRUTINA DE LA SEÑAL EN LA LINEA 100.
LINEA 20 ACTIVA LA DETECCION DE DICHA SEÑAL.
LINEA 30 SI PULSAS "s" FINALIZAS.
LINEA 40 IMPRIME "MSX".
LINEA 50 SALTA A LA LINEA 30.
LINEA 100 SONIDO MARCADOR (BEEP).
LINEA 110 LISTA UN MENSAJE.
LINEA 120 VUELVE AL PUNTO DE PARTIDA.



```
RUN
MSX
MSX
Pulsaste la barra espaciadora
MSX
```

PUNTOS A RECORDAR

Mientras el programa no se esté ejecutando dentro de las subrutinas de tratamiento de errores, no se detectarán interrupciones de este tipo.

PRECAUCIONES

Un error muy común es:

STRIG (0) ON ...

Está mal, ya que no debe haber ningún espacio entre STRIG y (0) *(Syntax error)*.

Si te equivocas en algún número de línea de la lista se producirá un error de número de línea no definido *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

STRIG( )
STRIG ON/OFF/STOP

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

OPEN

DESCRIPCION

Esta instrucción abre una determinada unidad y asocia con un *buffer* para dicha unidad; fija también el modo operativo para tal *buffer*, que es el intermediario para poder escribir y leer en la referida unidad. Antes de trabajar con un fichero tienes que abrirlo con la instrucción OPEN.

Has de ejecutar la instrucción OPEN antes de las siguientes instrucciones:

PRINT#	PRINT# USING
INPUT#	LINE INPUT#
INPUT$(#)	EOF

En esta versión MSX se pueden manejar cuatro unidades <descripción unidad>:

CAS: cassette.
CRT: pantalla modo texto.
GRP: pantalla modo gráfico.
LPT: impresora.

Cuando aparezcan la unidad de *floppy disc* y otros periféricos, habrá más unidades accesibles. Puedes ampliar las instrucciones de manejo de los periféricos con un cartucho de ROM. No olvides poner las comillas después de la descripción de la unidad.

Hay tres modos de E/S:

OUTPUT	especifica el modo de salida secuencial.
INPUT	especifica el modo de entrada secuencial.
APPEND	especifica el modo aditivo secuencial.

<número fichero> es un entero cuyo valor se encuentra comprendido entre 1 y MAXFILES —máximo número de ficheros—, que deben ir entre comillas en instrucciones como PRINT#, etc. El <número de fichero> se encuentra asociado a dicho fichero mientras que éste no se cierre (CLOSE).

SINTAXIS

OPEN "<descripción unidad>[<nombre fichero>]" [FOR <modo>] AS [#]<número fichero>.[<nombre fichero>],[FOR<modo>], y [#] son opcionales.

EJEMPLO

Para escribir caracteres en modo gráfico se ha de abrir un fichero para la unidad GRP. Podrás usar PRINT# para escribir en la pantalla en modo gráfico, en la posición que esté el cursor.


```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 DRAW "BM45,145"
40 PRINT #1,"ESCRITURA EN EL MODO 2"
50 GOTO 50
```

LINEA 10 ABRE UN FICHERO DE PANTALLA EN MODO GRAFICO.
LINEA 20 PONE LA PANTALLA EN MODO GRAFICO.
LINEA 30 LLEVA EL CURSOR AL PUNTO (30,145).
LINEA 40 ESCRIBE EN EL FICHERO 1.
LINEA 50 SE MANTIENE EN EL MODO GRAFICO.

PUNTOS A RECORDAR

El valor de MAXFILES es 1 en forma estándar, pero si estás trabajando con varios dispositivos en un programa no sería mala idea que le asignaras un valor más alto.

PRECAUCIONES

Si el número de fichero es mayor que el de MAXFILES tendrás un error en el número del fichero *(Bad file number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

MAXFILES
PRINT#
PRINT# USING
INPUT#
LINE INPUT#
INPUT$
VARPTR
EOF

Parte segunda, tema 42: "Gráficos avanzados (III): cómo imprimir caracteres en los modos gráficos de pantalla".

Parte segunda, tema 47: "Manejo de ficheros".

OR

DESCRIPCION

Se trata de uno de los operadores lógicos que sigue las reglas fundamentales del álgebra de Boole. También se emplea en las instrucciones IF/THEN/ELSE, con el objetivo de preguntar por más de una condición antes de la ejecución de cierto punto.

La tabla de verdad del operador lógico OR es:

OR	X	Y	X OR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1

Ejemplo: 34 OR 67

	34	0000000000100010
OR	67	0000000001000011
	99	0000000001100011

El operador OR se emplea en instrucciones del tipo IF/THEN/ELSE para tener condiciones múltiples.

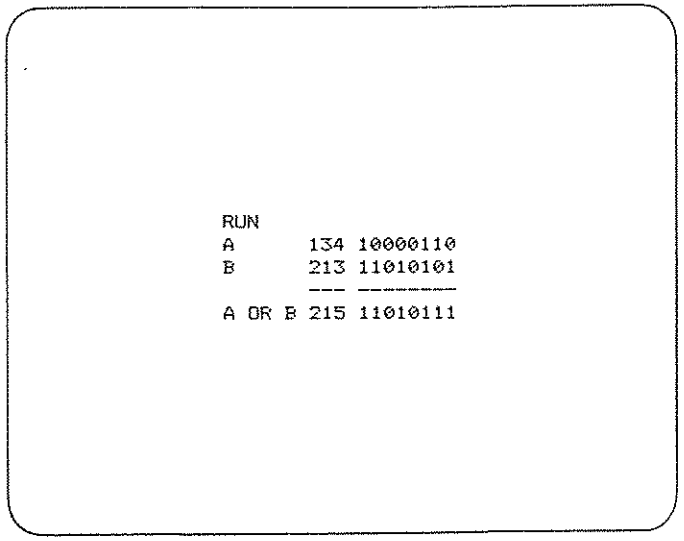
SINTAXIS

IF <condición> OR <condición> ... THEN ... ELSE ....
<var-num> = <número> OR <número>

EJEMPLO

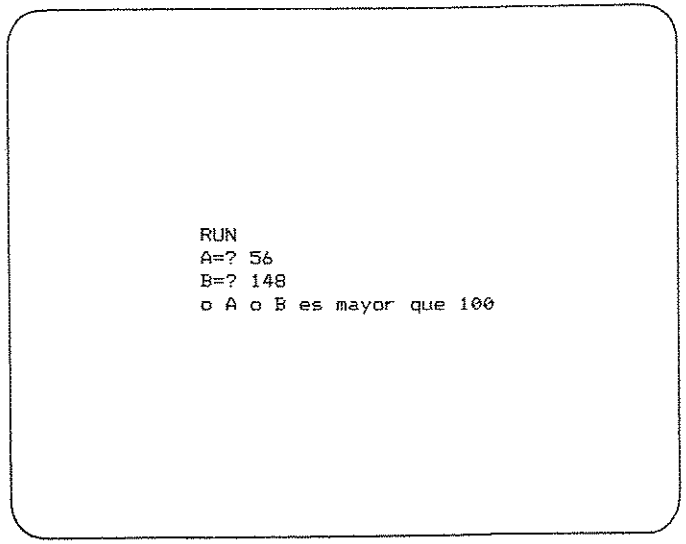
Algebra de Boole:

```
10 A=134
20 B=213
30 PRINT "A      ";A;BIN$(A)
40 PRINT "B      ";B;BIN$(B)
45 PRINT"        --- --------"
50 PRINT "A OR B";A OR B;BIN$(A OR B)
```

IF <condición> OR <condición> THEN

```
10 INPUT "A=";A
20 INPUT "B=";B
30 IF A>100 OR B>100 THEN PRINT "o A o B es mayor
que 100"
40 END
```



PUNTOS A RECORDAR

Puedes emplear el operador lógico OR para casar dos bits y crear, así, un valor en particular.

Los operandos con los que trabaja este operador, y todos los del álgebra de Boole, han de estar comprendidos entre −32768 y 32767.

Los operandos reales se truncan a enteros.

PRECAUCIONES

Si alguno de los operadores se sale del rango antes citado, se producirá un error de desbordamiento *(Overflow error)*.

Si alguno de los operandos fuese del tipo cadena de caracteres, se produciría un error en el tipo de datos *(Type mismatch error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

IF
THEN
ELSE
AND
NOT
IMP
XOR
EQV

Parte primera, tema 6: "Las condiciones".

Parte segunda, tema 34: "Algebra de Boole" (I).

Parte segunda, tema 35: "Algebra de Boole (II): la instrucción IF/THEN/ELSE".

OUT

DESCRIPCION

Esta instrucción envía a un determinado puerto un byte.

No emplees esta instrucción en *software* comercial, ya que depende de la máquina y perdería la compatibilidad con otros MSX.

SINTAXIS

OUT <puerto>,<dato>
<puerto> = tomará un valor entre 0 y 255
<dato> = es el byte enviado y tomará un valor entre 0 y 255.

EJEMPLO

OUT 1,116

PUNTOS A RECORDAR

Te aconsejamos el empleo de BIOS en vez de la instrucción OUT: de esta forma no perderás la compatibilidad con otros MSX.

PRECAUCIONES

Se producirá omisión de cualquier número negativo o mayor que 255.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

WAIT
INP

PAD

DESCRIPCION

El MSX puede llegar a tener conectados a los puertos para el *joystick* hasta dos lápices ópticos. Esta función devuelve el valor de la posición de la cabeza del lápiz.

SINTAXIS

PAD(<n>)
n = 0 a 3 para el lápiz óptico conectado al puerto 1 para el *joystick*.
PAD(1)= −1 si está en contacto.
0 si no está en contacto.
PAD(1)= coordenada X(0-255) del *pad* 1.
PAD(2)= coordenada Y(0-255) del *pad* 1.
PAD(3)= −1 si pulsas el botón del *pad*.
0 si no pulsas el botón el *pad*.
n=0 a 4 para el lápiz óptico conectado al puerto 2 para el *joystick*.
PAD(4)= 1 si está conectado.
0 si no está conectado.
PAD(5)= coordenada X(0-255) del *pad* 2.
PAD(6)= coordenada Y(0-255) del *pad* 2.
PAD(7)= −1 si pulsas el botón del *pad*.
0 si no pulsas el botón del *pad*.

EJEMPLO

Este pequeño programa emplea el lápiz óptico como una tiza de pizarra.

```
10 SCREEN 2,0,0
20 IF NOT PAD(0) THEN 20
30 X=PAD(1)
40 Y=INT(PAD(2)*192/255)
50 PSET X,Y
60 GOTO 20
```

LINEA 10 FIJA EL MODO GRAFICO DE ALTA RESOLUCION.
LINEA 20 SI NO PULSAS EL BOTON REPITE.
LINEA 30 PREGUNTA POR X.
LINEA 40 PREGUNTA POR Y TOMANDOLO SI CABE EN LA PANTALLA.
LINEA 50 PINTA UN PUNTO.

PUNTOS A RECORDAR

Observa que las coordenadas sólo son válidas cuando PAD(0) o PAD(4) valen −1. PAD(1) y PAD(2) toman un valor cuando PAD(0) está activado; PAD(3) y PAD(6) lo hacen al estar PAD(4) activado.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 51: "Periféricos".

PAINT

DESCRIPCION

La instrucción PAINT rellena un área delimitada por una línea con un determinado color.

Se han de especificar las coordenadas desde las que se quiere que el ordenador comience a dibujar. Si estás rellenando una determinada figura, indica a cualquier punto de la figura.

SINTAXIS

PAINT <especificación de coordenadas>

pinta con el color que en ese momento tiene asignado el texto.

PAINT <especificación de coordenadas>,<color>

pinta con el color especificado.

PAINT <especificación de coordenadas>,<color>,<color del borde>

pinta con el color especificado, bordeándolo con una línea del citado color (sólo en el modo multicolor).

<especificación de coordenadas>:

1) (<coordenada X>,<coordenada Y>)
   Rango de X = 0-255 Rango de Y = 0-191
2) STEP (<incremento X>,<incremento Y>)

Estas coordenadas se refieren al último punto al que apuntaba el ordenador.

Nota: <incremento X>, <incremento Y>, <coordenada X> y <coordenada Y> pueden ser <var-num>, <exp-num> o simplemente <const-num>.

CODIGOS DE COLOR

0	Transparente	8	Rojo (medio)
1	Negro	9	Rojo (claro)
2	Verde (medio)	10	Amarillo (oscuro)
3	Verde (claro)	11	Amarillo (claro)
5	Azul (claro)	12	Verde (oscuro)
6	Rojo (oscuro)	13	Magenta
7	Cian	15	Blanco

EJEMPLOS

1. En el modo gráfico de alta resolución (SCREEN 2).

En este modo el color del borde de la figura y el del texto han de ser iguales; en caso contrario el dibujo se extenderá. En este ejemplo no se especifica ningún color, de modo que usa los colores estándar, es decir, las rayas se dibujan en blanco sobre fondo azul y la figura se rellena de blanco.

```
10  SCREEN 2
20  PSET (100,100)
30  LINE-STEP(-50,50)
40  LINE-STEP(100,0)
50  LINE-STEP(50,-50)
60  LINE-STEP(-100,0)
70  PAINT (110,110)
100 GOTO 100
```

2. En el modo 3, modo gráfico de baja resolución, puedes dibujar los bordes de la figura de diferente color que el interior.

```
10  SCREEN 3
20  PSET (100,100)
30  LINE-STEP(-50,50)
40  LINE-STEP(100,0)
50  LINE-STEP(50,-50)
60  LINE-STEP(-100,0)
70  PAINT (110,110),9,15
100 GOTO 100
```

PUNTOS A RECORDAR

Sólo se puede especificar el color de la línea del borde en modo multicolor. Por tanto, sólo puedes especificar un color por cada figura que dibujes, es decir, no podrás tener las líneas que las bordean de distintos colores.

PRECAUCIONES

Si las coordenadas especificadas se salen de pantalla se producirá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DRAW
SCREEN

Parte primera, tema 27: "Pinta".

PDL

DESCRIPCION

PDL devuelve el valor del *paddle.*

Puedes conectar hasta doce juegos de *paddles,* seis en cada puerto para *joystick.*

SINTAXIS

PDL(<n>)
Puerto 1 n = 1,3,5,7,9,11
Puerto 2 n = 2,4,6,8,10,12

La función PDL devuelve un valor entre 0 y 255.

EJEMPLO

P1 = PDL(1)

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 51: "Periféricos".

PEEK

DESCRIPCION

Esta función devuelve el byte contenido en la posición de memoria especificada.

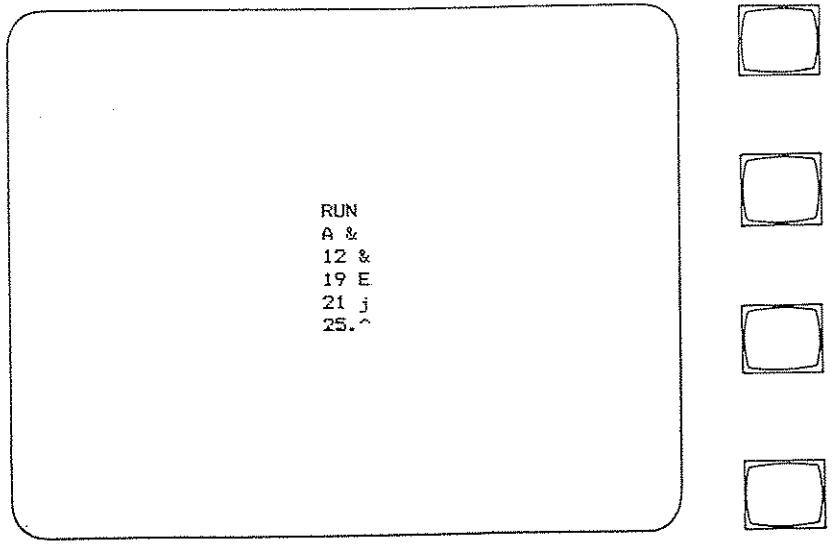
SINTAXIS

PEEK(<número entero>)

El rango es: (−32768, +65535).

EJEMPLOS

```
10 FOR I=1 TO 10000
20 A=PEEK(I)
30 IF A>31 AND A<127 THEN PRINT HEX$(I);" ";CHR$(A
)
40 NEXT I
```



PUNTOS A RECORDAR

POKE es la instrucción complementaria a PEEK.

Para preguntar por una posición de pantalla has de hacerlo con VPEEK.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

POKE
VPOKE
VPEEK

(Véase la tabla de variables del sistema.)
Parte segunda, tema 48: "Mapa de memoria".

PLAY

DESCRIPCION

Esta instrucción genera música en función del macrolenguaje musical, que se escribe en forma de cadenas de caracteres, al igual que el macrolenguaje gráfico.
El macrolenguaje musical:

A, B, C, D, E, F y G con las opciones #, +, −

Tocan la nota indicada en la octava en que se encuentra el ordenador.

o + después de la letra indica SOSTENIDO
− <menos> indica BEMOL.

Observa que los símbolos #, +, − sólo son válidos si esa nota existe, es decir, corresponde a una nota negra de piano.

O <n> n=1 ... 8	Selecciona la octava de 1 a 8. Cada octava varía de C a B. La octava que se selecciona por defecto es la cuatro; en dicha octava, C corresponde a la "C media" de un piano.
N <n> n=0 ... 96	Toca la nota *n*. Es otro modo de tocar cualquier nota teniendo así las ocho octavas disponibles.
L <n> n=1 ... 64	Fija la duración de la nota. L1 = nota entera (redonda). L2 = media nota (blanca). L4 = un cuarto de nota (negra). L8 = un octavo de nota (corchea). etcétera.

La duración de una nota en particular variará según sea el <n> que siga a la nota. Por ejemplo, L8C es lo mismo que C8. Este comando no se puede emplear con el comando N.
El valor prefijado de forma estándar es L4.

R<n> n=1 ... 64	Produce un silencio. La duración del silencio se fija del mismo modo que en el comando L.

R1 = pausa de una nota
R2 = pausa de media nota.
R4 = pausa de un cuarto de nta.
R8 = pausa de un octavo de nota.
etcétera.

Nota: R y R0 tienen el mismo efecto que la pausa que se fija del modo estándar y que vale R4.

.(punto)

El punto después de una nota de la A a la 6, N o la pausa R incrementa su longitud en 3/2 de lo normal, es decir, suena como una nota con punto.
Después de cada nota puedes incluir más de un punto.

T <n> n=32...255	TIEMPO El tiempo fija el número de cuartos de nota (o negras) que se tocan en un minuto. *n* ha de estar comprendido entre 32 y 255. Su valor estándar es 120.
V <n> n=0...15	Se trata del volumen. V fija el volumen de la música generada: *n* ha de estar comprendido entre 0 y 15. El volumen más alto es 15, y el más bajo es 0. El valor del volumen estándar es 8.
M <n> n=1...65535	Modulación. Fija el período de la envolvente especificada con S. El valor de la modulación estándar es 255.
S <n> n=0...15	Nos da una forma de envolvente. Hay varios tipos de ondas predefinidas. Véase SOUND para más detalles.
X <variable>	El comando X ejecuta el contenido de la variable de tipo cadena de caracteres que tiene como argumento; su efecto es como si se encontrase escrito directamente en el macrolenguaje musical.

Para todos los comandos anteriores, *n* puede ser una constante entera. o una variable; en este caso ha de ir expresado como "=<variable>;", es decir. la variable ha de ir precedida por "=" y añadir a su final ";".
Cuando ejecutes un BEEP (sonido) inicializas el sistema de sonido a los valores estándar.

SINTAXIS

PLAY <exp-caracteres para la voz 1>,<exp-caracteres para la voz 2>,<exp-caracteres para la voz 3>

Siendo las voces 2 y 3 opcionales.

EJEMPLO

```
10 A$="CDEFGAB"
20 FOR I=1 TO 8
30 PLAY "O=I;XA$;"
40 NEXT I
```

PUNTOS A RECORDAR

No se debe mezclar esta función con la instrucción PLAY( ), ya que son muy diferentes.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 28: "El macrolenguaje musical".

PLAY() (función)

DESCRIPCION

Esta función te indica si un determinado canal del ordenador se encuentra tocando música.

Devuelve:

−1 si todavía se encuentra tocando.
0 en caso contrario.

PLAY(0) analiza si los tres canales, 1, 2 y 3, se encuentran tocando música. En el caso de que alguno de ellos se encuentre tocando, devuelve el valor −1; en caso contrario devolverá el valor 0.

SINTAXIS

<var-num> = PLAY(<canal de sonido>)
<canal de sonido> = 0 ... 3

PUNTOS A RECORDAR

Esta función es completamente distinta a la instrucción PLAY.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PLAY

Parte primera, tema 28: "El macrolenguaje musical".

POINT

DESCRIPCION

Esta función indica el color de un determinado punto.
No devuelve el color de un *sprite*.

SINTAXIS

POINT (<coordenada X>,<coordenada Y>)

EJEMPLO

El siguiente programa te indica el color de las pantallas gráficas. En la línea 10 el ordenador lo imprime en la pantalla gráfica, mediante PRINT#1.

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 COLOR 14,8,9
40 CLS
50 P=POINT(100,100)
60 PRINT #1,"   EL CODIGO DE COLOR ES:";P
70 GOTO 70
```

Observarás cómo la pantalla se pone en color rojo, apareciendo el siguiente texto:

EL COLOR ES 8

PUNTOS A RECORDAR

Devuelve el valor 0 en los modos de texto 0 y 1.
Devuelve el valor −1 cuando las coordenadas especificadas como argumento se encuentran fuera de pantalla.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

COLOR
SCREEN
PSET
PRESET

Parte primera, tema 23: "Dibujar puntos".

POKE

DESCRIPCION

La instrucción POKE escribe un byte en una determinada posición de memoria.

POKE se emplea principalmente para insertar el código máquina en un programa BASIC. La parte más útil para grabar una subrutina en código máquina es el área de trabajo del usuario; esto se hace mediante la instrucción CLEAR.

SINTAXIS

POKE <dirección>,<expresión-entera>

<dirección> ha de estar comprendido entre −32768 y 65535.
Si es negativo, la dirección se calculará restándosela de 65535.
<expresión-entera> ha de ser un número que quepa en un byte; o sea, ha de estar comprendido entre 0 y 255.

EJEMPLO

POKE 65535,0

PUNTOS A RECORDAR

Ten cuidado con esta instrucción; puedes hacer caer el sistema, escribiendo en determinadas posiciones de memoria.

Para escribir en la RAM de la pantalla emplea VPOKE.

PRECAUCIONES

Si la <dirección> o la <expresión entera> se salen de sus rangos correspondientes entonces ocurrirá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CLEAR
PEEK
VPEEK
VPOKE

(Véase la tabla de variables del sistema.)

POS POSición del cursor

DESCRIPCION

Esta función devuelve la posición del cursor de texto. Necesitas una variable falsa para obtener la posición.

SINTAXIS

POS(0)

EJEMPLO

LET P=POS(0)

PUNTOS A RECORDAR

La posición del extremo izquierdo vale 0.
La posición del extremo derecho depende del modo de pantalla.

	ANCHO	
	estándar	máximo
modo de pantalla 0	37	40
modo de pantalla 1	29	32

PRECAUCIONES

Un error muy corriente es olvidarse de (0), la variable falsa.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

CRSLIN
WIDTH

Parte segunda, tema 40: "Gráficos avanzados (I): características de cada modo de pantalla".

PRESET borra un punto *(Point RESET)*

DESCRIPCION

Esta instrucción pone un punto al color que tiene el papel.

Si se especifica el color, pone el punto con el color especificado. (Es la misma instrucción que PSET.)

SINTAXIS

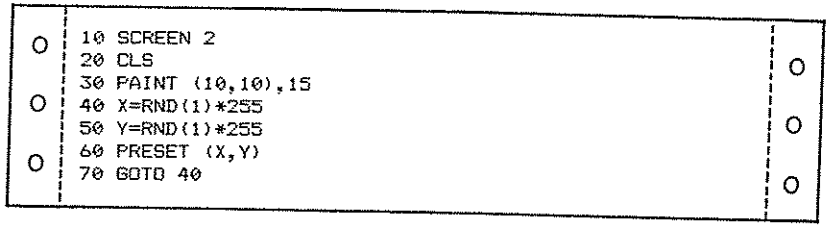
PRESET <especificación de coordenadas>
PRESET <especificación de coordenadas>,<color>

<especificación de coordenadas>

1) (<coordenada x>,<coordenada y>)
2) STEP (<incremento x>,<incremento y>)

<incremento x>, <incremento y>, <coordenada x>, <coordenada y> pueden ser <var-num>, <exp-num>, o simplemente <const-num>.

EJEMPLO



```
10 SCREEN 2
20 CLS
30 PAINT (10,10),15
40 X=RND(1)*255
50 Y=RND(1)*255
60 PRESET (X,Y)
70 GOTO 40
```

LINEA 10 MODO (2) DE ALTA RESOLUCION GRAFICA.
LINEA 20 PONE LA PANTALLA EN AZUL.
LINEA 30 PINTA DE BLANCO LA PANTALLA (COLOR DEL TEXTO).
LINEA 40 COORDENADA X.
LINEA 50 COORDENADA Y.
LINEA 60 CREA UN PUNTO CON EL COLOR DEL FONDO (AZUL).
LINEA 70 VUELVE ATRAS PARA CREAR MAS PUNTOS.

Verás la pantalla azul que se pinta de blanco y, entonces, la llena de montones de puntos azules de una forma aleatoria.

PUNTOS A RECORDAR

PRESET <especificación de coordenadas>, <color> tiene el mismo efecto que PSET <especificación de coordenada>, <color>.

PRECAUCIONES

La instrucción PRESET no hará nada si el punto está fuera de la pantalla. En cambio, si las coordenadas se salen del rango (−32768, 32767) habrá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PSET
POINT
COLOR

Parte primera, tema 23: "Dibujar puntos".

PRINT

DESCRIPCION

Es la instrucción del BASIC más usada. Escribe en la pantalla lo que tú le indiques.

A continuación de la instrucción PRINT puedes colocar una lista con todos los parámetros que quieras escribir; éstos han de ser números, variables numéricas o de tipo cadena de caracteres, o sencillamente cadenas de caracteres entre comillas. La posición en que te escribirá los parámetros se encuentra determinada por los símbolos de puntuación, o también por las instrucciones TAB y LOCATE.

El BASIC divide cada línea en zonas de catorce caracteres. La coma le indicará al ordenador que imprima cada parámetro al principio de cada una de estas zonas.

En cambio el punto y coma hará que los parámetros de la lista se escriban de manera secuencial dentro de la línea. La inclusión de uno o más espacios entre los parámetros tiene el mismo efecto que el punto y coma.

Si al final de una instrucción PRINT añades un punto y coma, producirá que la siguiente instrucción PRINT continúe escribiendo a partir de donde la otra lo dejó; esto es, no habrá un fin de línea.

Si al final de la lista de parámetros no pones nada, entonces el ordenador producirá un fin de línea, de modo que la siguiente instrucción PRINT comenzará en la siguiente.

En caso de que el ordenador no pudiera escribir todos los caracteres en una línea, continuaría en la siguiente.

Todos los números escritos irán precedidos y seguidos de un <espacio>; si el número fuera negativo, el signo menos aparecerá delante del número.

La abreviatura de PRINT es el símbolo de interrogación "?".

SINTAXIS

PRINT <lista de parámetros>
? <lista de parámetros>

EJEMPLO

```
10 PRINT "LOS MENSAJES HAN DE IR ENTRE COMILLAS"
20 PRINT "EL PUNTO Y COMA INDICA CONTINUACION  EN
LA MISMA LINEA ";
30 PRINT "DONDE LO DEJASTE"
40 A=100
50 B=300
60 PRINT "A=";A,"B=";B
70 PRINT "A+B=";A+B
```



```
RUN
LOS MENSAJES HAN DE IR ENTRE COMILLAS
EL PUNTO Y COMA INDICA CONTINUACION  EN LA MISMA
LINEA DONDE LO DEJASTE
A= 100         B= 300
A+B= 400
```

Este ejemplo necesita que la pantalla tenga una anchura (WIDTH) de veintinueve caracteres.

PUNTOS A RECORDAR

Observa que la abreviatura "?" de PRINT se transforma en esta última palabra cuando listas el programa.

Si lo que pretendes escribir es una tabla de valores te aconsejamos que emplees la instrucción PRINT USING.

PRECAUCIONES

No te olvides de encerrar entre comillas las cadenas de caracteres.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT USING
TAB
LOCATE

Parte primera, tema 2: "El modo directo".
Parte primera, tema 9: "Algo más sobre las salidas (PRINT) y la pantalla".

PRINT

DESCRIPCION

Para poder escribir datos en algunos periféricos, así como en las pantallas en modo gráfico, se ha de abrir un canal (OPEN) y emplear una instrucción especial, PRINT#. La almohadilla "#" debe ir seguida del número de fichero abierto con OPEN.

PRINT# se suele usar, también, para grabar ficheros en cassette.

SINTAXIS

PRINT# <número de fichero>

EJEMPLO

```
10 OPEN "GRP:" FOR OUTPUT AS #1
20 SCREEN 2
30 DRAW "BM45,100"
40 PRINT #1,"ESCRITURA EN MODO 2"
50 GOTO 50
```

LINEA 10 ABRE UN FICHERO EN LA PANTALLA GRAFICA COMO #1.
LINEA 20 SE PONE EN LA PANTALLA GRAFICA 2.
LINEA 30 MUEVE EL CURSOR GRAFICO AL PUNTO 30,100.
LINEA 40 ESCRIBE EN LA PANTALLA GRAFICA.
LINEA 50 SE MANTIENE EN ESA PANTALLA.

PUNTOS A RECORDAR

Has de abrir un canal (OPEN) antes de emplear la instrucción PRINT#. La instrucción INPUT# es complementaria a PRINT#.

PRECAUCIONES

Si el canal al que haces referencia no se encuentra abierto, se producirá un error en el número de fichero *(Bad file number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

MAXFILES
INPUT#
OPEN
CLOSE

Parte segunda, tema 42: "Gráficos avanzados (III): cómo escribir en las pantallas gráficas empleando ficheros".
Parte segunda, tema 47: "Manejo de ficheros".

PRINT USING

DESCRIPCION

Esta instrucción te permite escribir cadenas de caracteres y números con un formato específico. Te ayudará a tabular datos ordenadamente.

SINTAXIS

PRINT USING <exp-car>;<lista de datos>

<lista de datos> pueden ser números y cadenas de caracteres. Han de ir separados por punto y coma.

<car-exp> es una cadena con los caracteres que especifican el formato. Puede ser una cadena de caracteres o una variable de este tipo

EXPLICACION DE LOS CARACTERES

! Indica que sólo se imprima el primer carácter de cada dato.

```
F$="MSX": PRINT USING"!";F$
M
```

@ Inserta una determinada cadena en el punto que indica la @.

```
PRINT USING "ABC@DEF";MSX
ABCMSXDEF
```

El signo "#" indica que en esa posición se ha de escribir un dígito; por ejemplo: PRINT USING "#.###"; 1.3, que escribirá 1.300; es decir, formatea el número a escribir.

Como habrás visto en el ejemplo puedes incluir un punto decimal entre los signos. En caso de que el número a escribir tenga menos dígitos de los especificados, el número se ajustará a la derecha e irá precedido por los espacios necesarios.

```
PRINT USING "####.##";65.87
b65.87*
```

\+ Este signo ha de ir al principio o al final del formato, e indicará que en el lugar donde se ponga irá el signo "+" o "−" del número a escribir.

```
PRINT USING "+####.######"; −0.123422
−0.12342
PRINT USING "#####.#+";10.71
bb10.7+
```

* El símbolo b representa un espacio en blanco.

** Estos dos asteriscos indican que los espacios que haya en blanco se han de rellenar con asteriscos "*". Representan dos dígitos más, en los cuales podrás escribir, de ser necesario, un número.

```
PRINT USING "**#.##";5.55,−5.55
**5.55*−5.55
```

$$ Los dos símbolos dólar hacen que se imprima uno de éstos delante de cada número. Indican dos dígitos más, de los cuales uno de ellos siempre será $.

```
PRINT USING "$$#####.##";10000,99999.99,−100.55$10000.
005$99999.99−$100.55
```

No puedes combinar el formato para exponentes conjuntamente con $$.

**$ Esta especificación es una combinación de ** y $$. Todos los espacios en blanco se rellenarán con *, y el número irá precedido del símbolo $. Este formato añade tres posiciones más al número, uno de los cuales ha de ser $.

```
PRINT USING "**$####.##",34.99
***$34.99
```

, Si añades una coma a la izquierda del punto decimal dentro del formato hará que cada tres dígitos a la izquierda del punto decimal se escriba una coma *.

```
PRINT USING "#######,.##";2000000
2,000,000.00
```

Si la coma la sitúas al final del formato se escribirá una coma después de escribir el número.

```
PRINT USING "##.##,";12.567
12.56,
```

^^^^ Este formato es para el exponente, e indica que se ha de dejar espacio para poner E+XX. Con esta especificación también puedes indicar dónde irá el punto decimal; el número se ajustará a la izquierda y el exponente variará según dicho ajuste.

```
PRINT USING "##.##^^^^";200.00
b2.00E+02
```

* En la notación anglosajona la coma es nuestro punto, y su punto decimal es nuestra coma decimal.

EJEMPLO

```
PRINT USING "$$####,.";10000;2000.50;3000.555
$10,000.00 $2,000.50 $3,000.56
```

Tienes más ejemplos en la parte de *Guía avanzada de programación.*

PUNTOS A RECORDAR

Si el número a escribir no cabe en el formato especificado para la escritura, aparecerá un símbolo de tanto por ciento (%) delante del número. También te ocurrirá en caso de que el redondeo del número exceda el tamaño de su formato.

Por ejemplo:

```
PRINT USING "##.##";1000
%1000.000
PRINT USING "##.##";99.999
%100.00
```

PRECAUCIONES

Si el número de dígitos significativos en la especificación de formato excede de 24 se producirá un error de llamada ilegal a una función *(Illegal function call).*

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT# USING

Parte segunda, tema 36: "PRINT USING".

PRINT# USING

DESCRIPCION

Se trata de la misma instrucción que PRINT USING, con la salvedad de que ésta escribe en ficheros almacenados en cualquier tipo de periféricos. Antes de emplear esta instrucción se ha de abrir un canal para un periférico en concreto mediante la orden OPEN.

SINTAXIS

PRINT# <número de ficheros>, USING <exp-car>,<lista de datos>

EJEMPLO

```
PRINT#2, USING "##.####";3.453729
```

PUNTOS A RECORDAR

Antes de emplear la instrucción PRINT# USING debes abrir un canal mediante OPEN.

PRECAUCIONES

En caso de que hagas referencia a una canal que no se encuentre abierto, se producirá un error en el número de fichero *(Bad file number).*

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT USING
MAXFILES
PRINT#
OPEN
CLOSE

Parte segunda, tema 36: "PRINT USING".
Parte segunda, tema 41: "Gráficos avanzados (III): cómo escribir en las pantallas gráficas mediante ficheros".
Parte segunda, tema 47: "Manejo de ficheros".

PSET Fija un punto *(Point SET)*

DESCRIPCION

Marca un punto, en las coordenadas que le indiques, con el color actual u otro que se especifique.

SINTAXIS

PSET <especificación de coordenadas>
PSET <especificación de coordenadas>,<color>

Donde <especificación de coordenadas> puede ser:

1) (<coordenada X>,<coordenada Y>)
2) STEP (<incremento X>,<incremento Y>)

<incremento X>, <incremento Y>, <coordenada X>, <coordenada Y> y <color> pueden ser <var-num>, <exp-num> o simplemente <const-num>.

EJEMPLO

El siguiente programa te muestra un cielo negro lleno de estrellas multicolores.

```
10 COLOR 15,1,1
20 SCREEN 2
30 X=RND(1)*255
40 Y=RND(1)*255
50 Z=INT(RND(1)*16)
60 PSET (X,Y),Z
70 GOTO 30
```

LINEA 10 FONDO Y PAPEL DE COLOR NEGRO.
LINEA 50 COLOR ALEATORIO.

PUNTOS A RECORDAR

La instrucción PRESET especificando un color es exactamente la misma que PSET especificando, también, un color.

PRECAUCIONES

Si el punto que especificas está fuera de la pantalla, la instrucción PSET no hará nada; en cambio, si las coordenadas del punto exceden del rango de enteros permitido, se producirá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

COLOR
PRESET

Parte primera, tema 23: "Dibujar puntos".

PUT SPRITE

DESCRIPCION

Esta instrucción sitúa un *sprite* en pantalla. Sólo se te permite uno por cada plano de *sprites*. Hay 32 planos de éstos; por tanto, podrás llegar a tener como máximo 32 *sprites* en pantalla.

La especificación de coordenadas indica al ordenador donde situar al *sprite*. Puedes, incluso, esconderlo en los límites de la pantalla, o pasarlo por debajo o sobre otro *sprite*.

Los *sprites* pueden tener cualquiera de los dieciséis colores, pero sólo uno por *sprite*.

Consulta el tema de *sprites* en la *Guía avanzada de programación*, con el objetivo de que sepas más sobre su manejo.

SINTAXIS

PUT SPRITE <número de plano de *sprite*>[<especificación de coordenadas>][,<color>][,<número de modelo>].

Todos los argumentos pueden ser <const-num>, <var-num> o <exp-num>, pero han de estar dentro de su rango correspondiente.

<número de plano de *sprite*> = de 0 a 31

<especificación de coordenadas>

1) (<coordenada X>,<coordenada Y>)
2) STEP (<incremento X>,<incremento Y>)

Rango de coordenadas:

entre −32768 y 32767

Rango de coordenadas en pantalla:

X = (−32,255)
Y = (−32,191)

Si las coordenadas se salen de su rango en pantalla aparecerán automáticamente por el lado opuesto. (Véase la *Guía avanzada de programación.*)

<número de modelo> = (0,256) para *sprites* de 8 × 8 *pixels.*
= (0,64) para *sprites* de 16 × 16 *pixels.*

EJEMPLO

```
10 SCREEN 2
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 X=100 : Y=120
40 C=7
50 PUT SPRITE 0,(X,Y),C,0
60 GOTO 60
```

LINEA 10 PONE LA PANTALLA EN MODO 2.
LINEA 20 DEFINE EL SPRITE (0) COMO UN CUADRADO.
LINEA 30 COORDENADAS X e Y.
LINEA 40 FIJA EL COLOR CIAN (7).
LINEA 50 SITUA EL MODELO DE SPRITE 0 EN EL PLANO 0 EN LAS COORDENADAS (X,Y) CON COLOR CIAN.

PUNTOS A RECORDAR

Cuando a la coordenada Y se le dan los valores 208 (&HD0) o 209 (&HD1) esconderá temporalmente uno o más *sprites*. (Véase la *Guía avanzada de programación:* parte de *sprites*, para más detalles.)

No podrás emplear los *sprites* estando en el modo de texto 0.

El tamaño del *sprite* está condicionado por la instrucción SCREEN.

En caso de que no especifiques las coordenadas de situación del *sprite*, el ordenador lo colocará en el último punto al que hizo referencia.

PRECAUCIONES

Si alguna coordenada se sale del rango de los enteros se producirá un error de desbordamiento *(Overflow error).*

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON SPRITE GOSUB
SPRITE$
SPRITE ON/OFF/STOP
SCREEN
COLOR

Parte primera, tema 15: "Gráficos avanzados (IV): los *sprites*".

READ

DESCRIPCION

Esta instrucción lee datos de la sentencia DATA y se los asigna a la variable especificada mediante READ.

Las instrucciones READ y DATA deben emplearse siempre conjuntamente. Los datos se almacenan en las líneas DATA, que son instrucciones no ejecutables; su único propósito es el de almacenar datos para el programa. Para poder acceder a esta información debes emplear la instrucción READ.

Puedes leer tantos variables como desees con una única instrucción READ; éstas pueden ser elementos de matrices, de tipo cadena de caracteres o numéricas; pero teniendo en cuenta que el tipo de dato y de la variable coincidan. Si no ocurriese así se produciría un error de sintaxis *(Syntax error)*.

La instrucción READ comienza a leer la información desde la primera sentencia DATA y continúa desde ésta con la siguiente. En caso de que quisieras leer de una determinada sentencia DATA deberás emplear la orden RESTORE, posicionándote así en la línea especificada, y entonces proceder a la lectura.

SINTAXIS

READ <lista de variables>

EJEMPLO

```
10 READ A$,B
20 PRINT A$;B
30 READ C$,D
40 PRINT C$;D
50 DATA "ASTRONOMIA",500,"FISICA",600
```



```
RUN
ASTRONOMIA 500
FISICA 600
```

PUNTOS A RECORDAR

Como puedes ver en el ejemplo anterior, el ordenador memoriza el último dato leído y, con la siguiente instrucción READ, procede a leer el siguiente dato no leído después del que sí lo ha sido.

PRECAUCIONES

En caso de que no coincidan los tipos de la variable y del dato leído se producirá un error de sintaxis *(Syntax error)*.

Si intentas leer más datos y ya han sido todos leídos, entonces se producirá un error de fin de datos *(Out of DATA)*. La solución sería emplear una orden RESTORE.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DATA
RESTORE

Parte primera, tema 12: "Lectura de datos de matrices".

REM

DESCRIPCION

Esta instrucción te permite introducir comentarios dentro de tu programa. Son instrucciones no ejecutables; cuando el ordenador se encuentra con una de ellas se la salta y continúa ejecutando la siguiente.

Se emplean para aclaraciones del programa. Un buen método es indicar qué hace cada sección, ya que de lo contrario resultará difícil entender tu programa. Si empleas inteligentemente estas instrucciones podrás ver siempre qué hace; de esta manera lo harás más inteligible.

Estas instrucciones también se deben emplear en los pequeños programas.

La abreviatura de REM es el apóstrofo.

SINTAXIS

REM comentario...
' comentario...

EJEMPLO

```
10 REM Mi primer programa
20 REM Titulo  : OVNI
30 REM Fecha   : 18/6/85
40 REM Version : 1
50 REM
..
..
..
RESTO DEL PROGRAMA
```

PUNTOS A RECORDAR

Puedes añadir REM al final de una línea (:REM...), pero no se te ocurra hacerlo a continuación de una DATA, ya que el ordenador los consideraría más datos.

Estas sentencias hacen más intelegible el programa, pero también están ocupando un precioso espacio de memoria. En caso de que te sea preciso utilizar más memoria para un programa, procede a borrar (DELETE) todos los REM que sean necesarios.

No hagas saltos (GOTO, GOSUB) a sentencias REM. Si tuvieras que borrar ese comentario se produciría un error de salto a una línea no definida *(Undefined line number)* al ejecutar el programa.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 3: "Escritura de un programa".

RENUM· RENUMera

DESCRIPCION

Con esta instrucción se renumeran las líneas del programa. Es frecuente que te cause trastornos el que las líneas de tu programa no estén ordenadas con un incremento fijo. Emplea RENUM: es el mejor modo de dar a tu programa un aspecto más decente, y te facilitará la inserción de nuevas líneas.

La orden RENUM por sí sola renumera todas las líneas comenzando desde la 10 y con incrementos sucesivos de 10 en 10; ésta es la estructura de líneas más usada, pero podrás renumerar comenzando desde la línea que quieras y con el incremento que desees.

RENUM cambia automáticamente todos los números de línea relacionados con las instrucciones GOTO, GOSUB, RESTORE, THEN, ELSE, ON GOTO, ON GOSUB.

SINTAXIS

RENUM

Renumera desde 10 con incrementos de 10.

RENUM <línea1>

Renumera desde la <línea1> con incrementos de 10.

RENUM <línea1>,<línea2>.

Renumera desde la antigua <línea2>, dando el valor de <línea1> y con un incremento de 10.

RENUM <línea1>,<línea2>,<incremento>

Renumera desde la antigua <línea2>, dándole el valor de <línea1> y con el incremento especificado.

RENUM ,<línea2>,<incremento>

Renumera desde la antigua <línea2>, dándole el valor 10 y con el incremento especificado.

RENUM ,<línea2>.

Renumera desde la antigua <línea2>, dándole el valor 10 y con incremento de 10.

RENUM „<incremento>

Renumera desde la antigua línea número 10 y con el incremento especificado.

RENUM <línea1>„<incremento>

Renumera desde la antigua línea 10, dándole el valor de <línea1> y con el incremento especificado.

EJEMPLOS

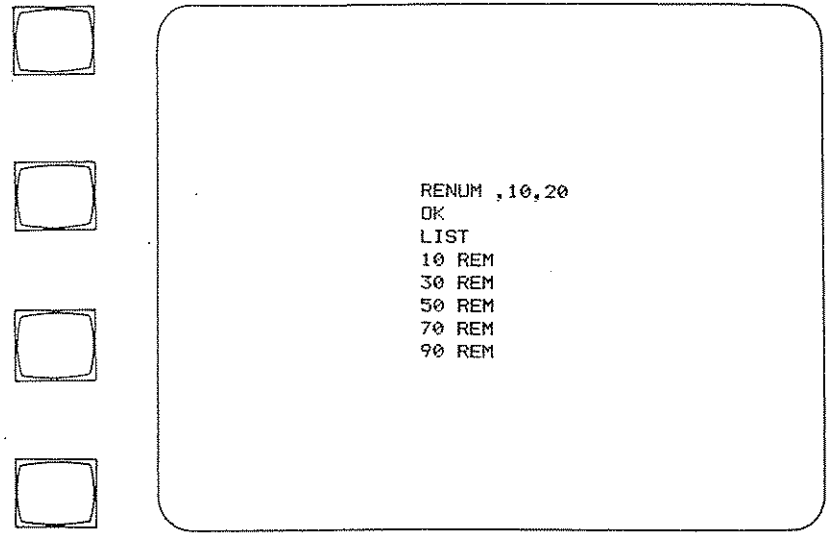
RENUM „100

Renumera desde la línea 10 incrementándose de 100 en 100.

RENUM 1000,200,20.

Renumera desde la antigua línea 200, comenzando con el nuevo número de línea 1000, incrementándose de 20 en 20.

```
10 REM
20 REM
30 REM
40 REM
50 REM
```



```
RENUM ,10,20
OK
LIST
10 REM
30 REM
50 REM
70 REM
90 REM
```

PRECAUCIONES

Si hubiese una referencia a una línea no existente en una instrucción GOTO, GOSUB o cualquier otra que haga dicho tipo, se producirá un error con el mensaje "*Undefined line NNNN in MMMM*", es decir, que en la línea MMMM se hace referencia a la línea NNNN, la cual no existe.

Por ejemplo:

```
10      GOTO 97
RENUM
```

Undefined line 97 in 10 (la línea 10 hace referencia a la línea no existente 97).

Los descuidos con esta instrucción no se ejecutarán; por ejemplo, RENUM 10,40 no tendrá ningún efecto si los antiguos números de línea fuesen 10,20,30. El resultado de estos descuidos será un error de llamada ilegal a una función *(Illegal function call)*.

Los incrementos negativos no se permiten.

No puede haber números de línea superiores a 65535; en caso de que durante la ejecución de esta orden se supere dicha cifra se producirá un error de llamada ilegal a una función.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

Parte segunda, tema 29: "Edición avanzada de programas".

RESTORE

DESCRIPCION

Realmacena instrucciones DATA. Cuando el ordenador lee (READ) datos de las sentencias DATA comienza en aquella que tiene menor número de línea y continúa con las siguientes hasta que terminan los datos. Si deseas volver a leer una determinada línea de datos, emplea la instrucción RESTORE apuntando a la línea de datos requerida.

Después de ejecutarse esta instrucción, la siguiente orden READ leerá el primer dato de la línea de datos apuntada. En caso de que no especifiques la línea de datos se tomará la sentencia DATA con menor número de línea.

SINTAXIS

RESTORE
RESTORE <línea>

EJEMPLO

```
10 READ A$
20 PRINT A$
30 RESTORE
40 READ B$
50 PRINT B$
60 DATA "EDAD DE ORO"
```

```
RUN
EDAD DE ORO
EDAD DE ORO
```


PUNTOS A RECORDAR

Emplea esta instrucción para evitar los errores de intento de lectura tras el fin de datos *(Out of data)*.

PRECAUCIONES

Si la orden RESTORE apunta a una línea no existente, se producirá un error de línea no definida *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

READ
DATA

Parte primera, tema 12: "Lectura de datos de matrices".

RESUME

DESCRIPCION

Esta instrucción reanuda la ejecución del BASIC después de ejecutar la subrutina de tratamiento de errores apuntados por ON ERROR GOTO. Después de haber tratado el error, la orden RESUME indica al ordenador que continúe la ejecución del programa en el punto que le hayas indicado.

Si no incluyes esta instrucción dentro de la subrutina de tratamiento de errores se producitá un error sin reanudación del programa, excepto si detienes la ejecución incluyendo STOP o END dentro de dicha subrutina.

Aquí tienes tres modos de emplear RESUME.

SINTAXIS

RESUME o RESUME0

continúa la ejecución desde la línea que ocasionó el error.

RESUME NEXT

continúa la ejecución desde la línea siguiente a la que ocasionó el error.

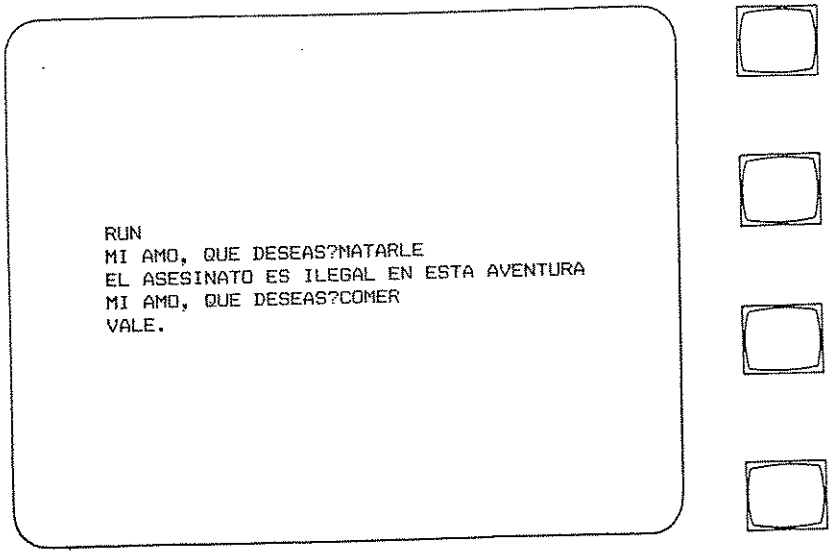
RESUME <línea>

continúa la ejecución desde la línea especificada.

EJEMPLO

Esta subrutina trata todos los comandos que contiene la palabra MATAR como errores (255) en la subrutina de tratamiento de éstos; se ayuda de la función ERR. RESUME 20 indica al ordenador que continúe la ejecución en la línea 20 después de mostrarte un mensaje.

```
10   ON ERROR GOTO 1000
20   INPUT "MI AMO, QUE DESEAS";A$
30   IF INSTR(A$,"MATAR") THEN ERROR 255
40   PRINT "VALE." : END
1000 IF ERR=255 THEN PRINT "EL ASESINATO ES ILEGAL
 EN ESTA AVENTURA" : RESUME 20
1010 END
```

PUNTOS A RECORDAR

No puedes emplear la orden RESUME para acceder al BASIC desde el modo directo.

PRECAUCIONES

Si existiese una instrucción RESUME fuera de la subrutina de tratamiento de errores ocasionará un error de RESUME sin error correspondiente *(RESUME without error)*.

Si dicha orden hace referencia a una línea no existente se producirá un error de número de línea no definido *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON ERROR GOTO
ERL
ERR
ERROR

Parte segunda, tema 38: "Tratamiento de errores".

RETURN

DESCRIPCION

Vuelta (RETURN) de una subrutina.

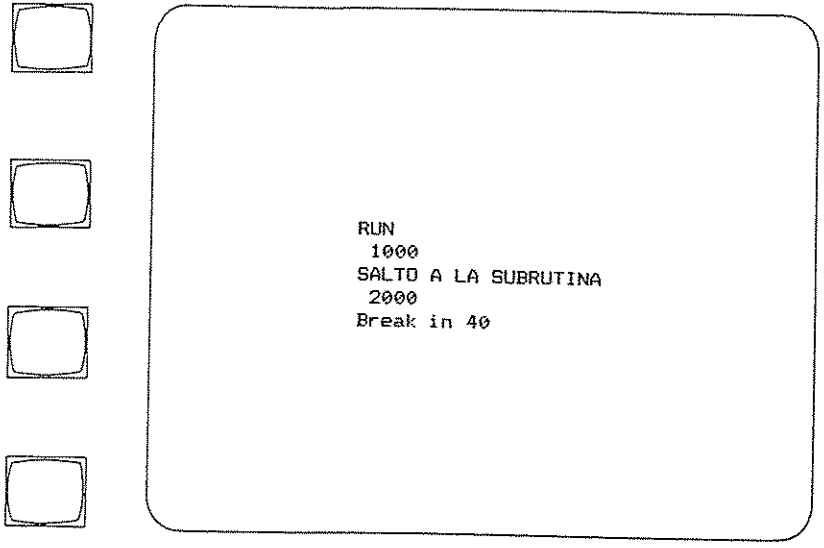
Esta orden hace que el ordenador continúe con la siguiente instrucción que sigue al GOSUB que hizo referencia a la subrutina en la que estamos.

SINTAXIS

RETURN

EJEMPLO

```
10 PRINT 1000
20 GOSUB 50
30 PRINT 2000
40 STOP
50 PRINT "SALTO A LA SUBRUTINA"
60 RETURN
```



PUNTOS A RECORDAR

Es bastante fácil entrar a ejecutar una subrutina después del programa principal; para evitar esto colocamos una instrucción STOP o END al final del programa principal.

Si la lógica de tu programa requiere más de una orden RETURN dentro de una subrutina, ponla: está totalmente permitido.

Esta instrucción también se emplea en las subrutinas de tratamiento de todo tipo de interrupciones.

PRECAUCIONES

Si el ordenador encuentra una orden RETURN fuera de una subrutina, o sea, sin previa llamada GOSUB, se producirá un error de RETURN sin GOSUB *(RETURN without GOSUB)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

GOSUB
ON ... GOSUB
ON ERROR GOSUB
ON INTERVAL GOSUB
ON KEY GOSUB
ON SPRITE GOSUB
ON STOP GOSUB
ON STRIG GOSUB

Parte primera, tema 17: "Estructura de tus programas".

RIGHT$

DESCRIPCION

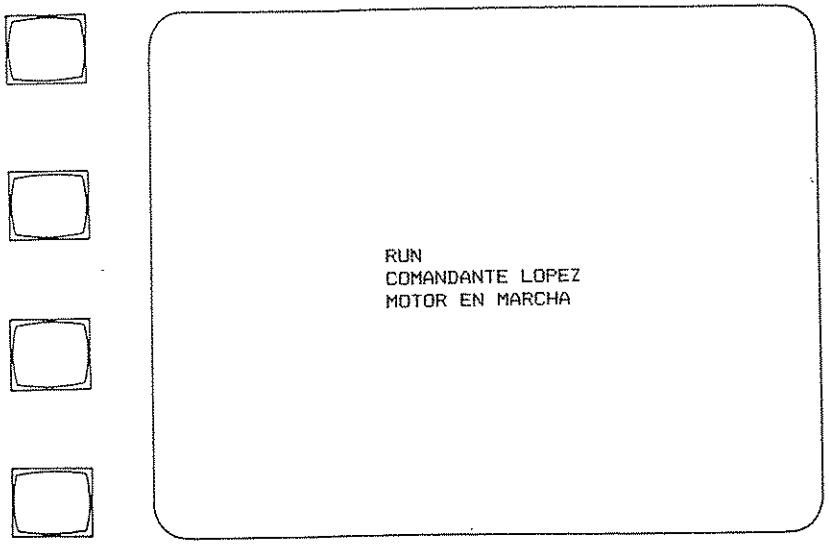
Esta es una función que maneja cadenas de caracteres. RIGHT$(A$,n) te devuelve los *n* caracteres más a la derecha de la cadena A$. Si *n* es igual a la longitud de la cadena obtendrás la misma cadena. Si *n* fuese cero, entonces la función devolvería una cadena vacía.

SINTAXIS

RIGHT$(<exp-car>,<número entero>)

EJEMPLO

```
10 A$="TORRE DE CONTROL A COMANDANTE LOPEZ"
20 PRINT RIGHT$(A$,16)
30 B$="COMIENZA LA CUENTA ATRAS, MOTOR EN MARCHA"
40 C=15
50 D$=RIGHT$(B$,C)
60 PRINT D$
```



PRECAUCIONES

El argumento de esta función es una cadena de caracteres seguida de un número entero; en caso de que estuviesen en otro orden, se produciría un error en el tipo de datos *(Type mismatch error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

LEFT$
MID$
INSTR

Parte primera, tema 14: "Las cadenas de caracteres".

RND Número aleatorio *(RaNDom number)*

DESCRIPCION

La función RND genera un número aleatorio entre 0 y 1. Cada vez que ejecutes el programa (RUN) se producirán las mismas series numéricas aleatorias. Pero si deseas que no se generen estas series de forma idéntica tienes que hacer RND(—TIME).

SINTAXIS

RND(<número>)

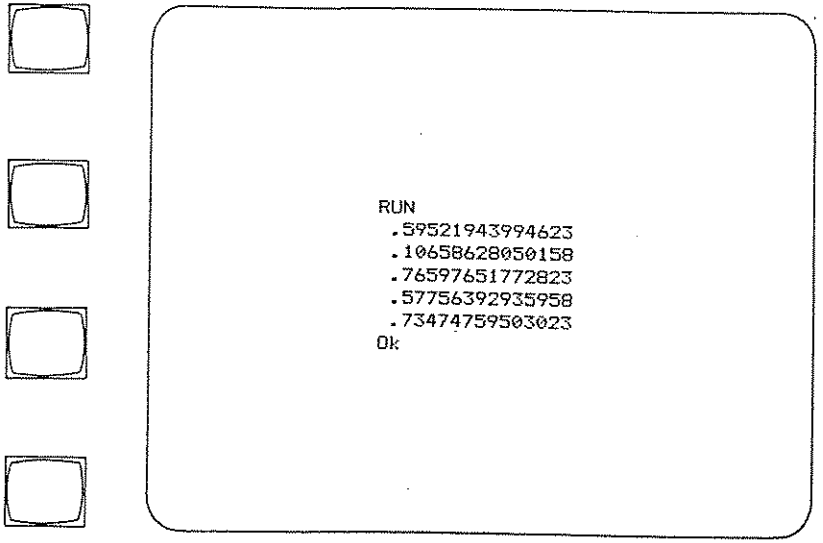
Si el <número> es negativo hará que esta función genere diferentes series, según el valor del <número>.

Si el <número> es 0, entonces la función devolverá el mismo valor que en la vez anterior.

Si el <número> es positivo se generará un número más de la secuencia.

EJEMPLO

```
10 FOR I=1 TO 5
20 PRINT RND(1)
30 NEXT I
```



```
RUN
 .59521943994623
 .10658628050158
 .76597651772823
 .57756392935958
 .73474759503023
Ok
```


Nota: Este programa generará la misma serie de números aleatorios cada vez que se ejecute.

Si quieres un número aleatorio entre 0 y 9 emplea esta función:

```
X=INT(RND(1)*10)
```

PUNTOS A RECORDAR

Los números que genera esta función tienen catorce dígitos, son de doble precisión y están comprendidos entre 0 y 0.99999999999999.

RND(—TIME) te dará un auténtico número aleatorio; sin embargo, RND(1) te dará siempre la misma serie de números.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 15: "Funciones".

RUN

DESCRIPCION

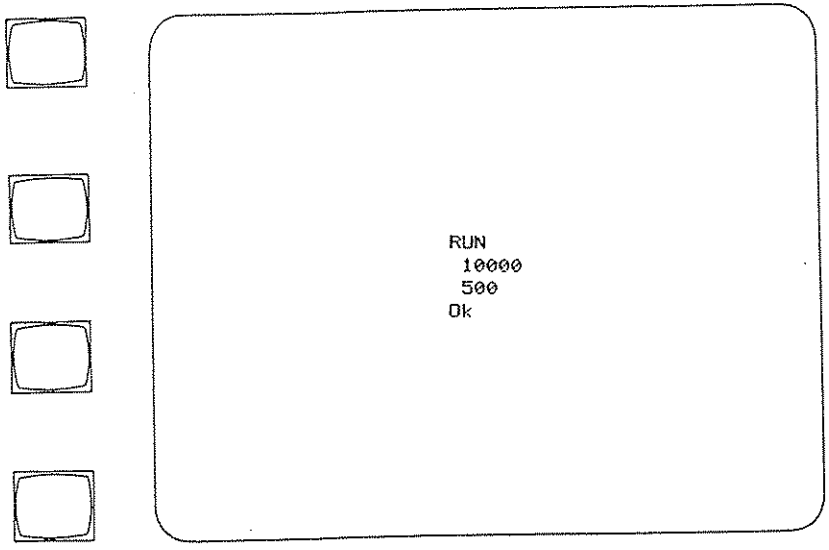
Ejecuta un programa.

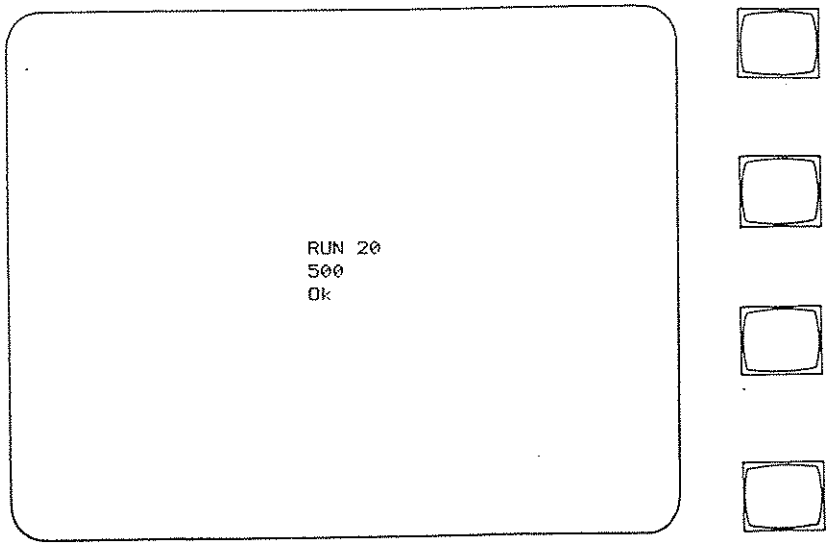
SINTAXIS

RUN
RUN <línea> ... Ejecuta el programa a partir de la línea indicada.

EJEMPLO

```
10 PRINT 10000
20 A=300
30 B=200
40 PRINT A+B
```





PUNTOS A RECORDAR

¿Cómo detener un programa? END y STOP detendrán la ejecución de tu programa.

<CTRL><STOP> detiene la ejecución desde el teclado.

Cuando conectes el ordenador la tecla de función F5 está programada para ejecutar (RUN).

PRECAUCIONES

Si no tienes ningún programa BASIC en memoria, esta orden hará que se muestre en pantalla "OK".

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

END
STOP

Parte primera, tema 3: "Escritura de un programa".
Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

SAVE

DESCRIPCION

Graba un programa BASIC en una determinada unidad, con formato de fichero ASCII.

Este comando se emplea para unir dos programas. (Véase MERGE y el tema del manejo del cassette en la *Guía avanzada de programación.)*

SINTAXIS

SAVE "<nombre-programa>"
SAVE "<descriptor-unidad>[<nombre fichero>]"

<descriptor-unidad> = CAS: para el cassette. De momento el MSX no tiene más unidades.

EJEMPLO

Supongamos que este programa está en memoria.

```
10 PRINT "HOLA"
20 PRINT "BIENVENIDO"
30 PRINT "AL"
50 FOR I=1 TO 10
60 PRINT "MSX"
70 NEXT I
```

Para grabarlo en formato ASCII con el nombre "PROG" teclea:

SAVE "PROG" o SAVE "CAS:PROG"

Para volverlo a cargar en el ordenador teclea:

LOAD "PROG" o LOAD "CAS:PROG"

PUNTOS A RECORDAR

Esta instrucción es distinta a CSAVE; SAVE graba los ficheros en formato ASCII mientras que CSAVE los graba en formato de indexado.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

MERGE
LOAD

Parte segunda, tema 39: "Grabación y carga con el cassette".

SCREEN

DESCRIPCION

SCREEN realiza diversas funciones; se emplea para fijar varios modos de pantalla, cassette, teclado, *sprite* e impresora.

Generalmente se emplea para fijar los modos de pantalla.

Mediante esta instrucción puedes determinar el tamaño de los *sprites*, pero sólo puedes tener un determinado tamaño en cada instante.

También fija la velocidad de transmisión del cassette, el *switch* de tecla y selección de impresora.

SINTAXIS

SCREEN [<modo>][,<tamaño *sprite*>][,<*switch* tecla>][,<transmisión cassette>][,<opción impresora>]

<modo> = 0,1,2,3	EJEMPLOS
0 = 40 × 24 modo texto	SCREEN 0
1 = 32 × 24 modo texto	SCREEN 1
2 = modo de alta resolución	SCREEN 2
3 = modo multicolor	SCREEN 3

<tamaño sprite> = 0,1,2,3

0 = 8 × 8 reducido	SCREEN ,0
1 = 8 × 8 ampliado	SCREEN ,1
2 = 16 × 16 reducido	SCREEN ,2
3 = 16 × 16 ampliado	SCREEN ,3

Nota: Cuando se especifique el tamaño de *sprite* se borrará el contenido de SPRITE$.

<switch tecla> = 0, distinto de cero

0 = desactiva el *switch* de tecla	SCREEN ,,0
distinto de cero = activa el *switch* de tecla	SCREEN ,,1

<velocidad transmisión cassette en baudios> = 0,1

0 = 1.200 baudios	SCREEN ,,,0
1 = 2.400 baudios	SCREEN ,,,1

Nota: La velocidad de transmisión puede cambiarse también con CSAVE.

<opción impresora> = 0, distinto de cero

0 = impresora MSX, con su capacidad gráfica	SCREEN ,,,,0
distinto de 0 = impresora no compatible con MSX; los caracteres gráficos se cambian a espacios	SCREEN ,,,,1

EJEMPLO

Este programa te muestra las diferencias entre el modo 2 (alta resolución) y el modo 3 (baja resolución). Lleva a cabo el mismo trabajo en ambas pantallas, mostrándote así las ventajas y desventajas de cada modo.

```
10  SCREEN 2
20  GOSUB 90
30  PAINT (105,120)
40  FOR I=1 TO 1000 : NEXT
50  SCREEN 3
60  GOSUB 90
70  PAINT (105,120),8,15
80  GOTO 80
90  PSET (100,100)
100 DRAW "R50D50L50U50F50"
110 RETURN
```

LINEA 10 MODO GRAFICO DE ALTA RESOLUCION.
LINEA 20 SALTO A LA SUBRUTINA.
LINEA 30 PINTA CON EL COLOR ACTUAL DE FONDO.
LINEA 40 RETARDO.
LINEA 50 MODO GRAFICO MULTICOLOR.
LINEA 60 SALTO A LA SUBRUTINA.
LINEA 70 COLOREA DE ROJO EL AREA DELIMITADA POR LA LINEA.
LINEA 90 SUBRUTINA: POSICIONA EL CURSOR GRAFICO.
LINEA 100 DIBUJA UN CUADRADO CON UNA DIAGONAL.
LINEA 110 VUELVE.

PUNTOS A RECORDAR

Las características de cada modo de pantalla las tienes en el tema de "Gráficos avanzados de pantalla".

PRECAUCIONES

Si utilizas la instrucción INPUT en los modos 2 y 3 ocasionará que la pantalla vuelve al último modo texto de pantalla empleado.

Si empleas cualquiera de las instrucciones gráficas en modo texto, excepto PUT SPRITE en modo 1, ocasionarás un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 9: "Algo más sobre las salidas (PRINT) y la pantalla".
Parte primera, tema 21: "Modos de pantalla".
Parte segunda, tema 39: "Grabación y carga con el cassette".
Parte segunda, tema 40: "Gráficos avanzados (I): características de cada modo de pantalla".
Parte segunda, tema 43: "Gráficos avanzados (IV): los *sprites*".
Parte segunda, tema 51: "Periféricos".

SGN SiGNo

DESCRIPCION

Esta función devuelve el signo del parámetro dado:

−1 si el parámetro es negativo.
0 si el parámetro es cero.
1 si el parámetro es positivo.

SINTAXIS

SGN(<número>)

EJEMPLO

```
PRINT SGN(-1000)
-1
```

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ABS

Parte primera, tema 15: "Funciones".

SIN seno *(SINe)*

DESCRIPCION

La función SIN devuelve el valor del seno del argumento asignado, que debe ir en radianes.

SINTAXIS

SIN(<número>)

EJEMPLO

```
PRINT SIN(1)
.84147098480792
```

PUNTOS A RECORDAR

El valor del seno se calcula en doble precisión, teniendo así el resultado catorce dígitos significativos.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

COS
TAN
ATN

Parte primera, tema 19: "Funciones matemáticas".

SOUND

DESCRIPCION

Esta instrucción escribe lo que le indique directamente en el *chip* generador de sonido programable (PSG). Este *chip* tiene catorce registros. Si quieres una explicación más completa de esta instrucción la obtendrás consultando el tema de "Efectos de sonido con el PSG".

REGISTRO	RANGO	DESCRIPCION
0	0-255	Ajuste fino de la frecuencia (canal A)
1	0-15	Ajuste clásico de la frecuencia (canal A)
2	0-255	Ajuste fino de la frecuencia (canal B)
3	0-15	Ajuste clásico de la frecuencia (canal B)
4	0-255	Ajuste fino de la frecuencia (canal C)
5	0-15	Ajuste clásico de la frecuencia (canal C)
6	0-31	Frecuencia del generador de ruido
7	0-63	Mezclador (cada bit tiene un control específico)
	bit 0	Sonido del canal A 0 = ON 1 = OFF
	bit 1	Sonido del canal B 0 = ON 1 = OFF
	bit 2	Sonido del canal C 0 = ON 1 = OFF
	bit 3	Ruido del canal A 0 = ON 1 = OFF
	bit 4	Ruido del canal B 0 = ON 1 = OFF
	bit 5	Ruido del canal C 0 = ON 1 = OFF
8	0-16	Control del volumen del canal A emplea una envolvente si su valor es 16
9	0-16	Control del volumen del canal B emplea una envolvente si su valor es 16
10	0-16	Control del volumen del canal C emplea una envolvente si su valor es 16
11	0-255	Período de la envolvente (bajo)
12	0-255	Período de la envolvente (alto)
	Período de la envolvente = (0-65535) = R12*256 + R11	

Registro	envolvente número	Forma de envolvente
13	0-15	
	0,1,2,3,9	
	4,5,6,7,15	

8

10

11

12

13

14

SINTAXIS

SOUND<registro>,<dato>

EJEMPLO

Efecto especial de sonido.

```
10 SOUND 7,62
20 SOUND 1,0
30 SOUND 0,254
40 SOUND 8,16
50 SOUND 13,9
60 SOUND 12,60
70 SOUND 11,0
```


PRECAUCIONES

Si no escuchas nada puede ser que tengas un error sintáctico, o bien que el volumen de tu televisor esté totalmente al mínimo.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PLAY

Parte segunda, tema 46: "Efectos de sonido con el PSG".

SPACE$ Cadena de espacios

DESCRIPCION

Esta función devuelve una cadena de caracteres <espacio>, tantos como le indiques en el argumento. Este puede ser un número real, pero ha de estar comprendido entre 0 y 255. La parte fraccionaria del número real se desprecia.

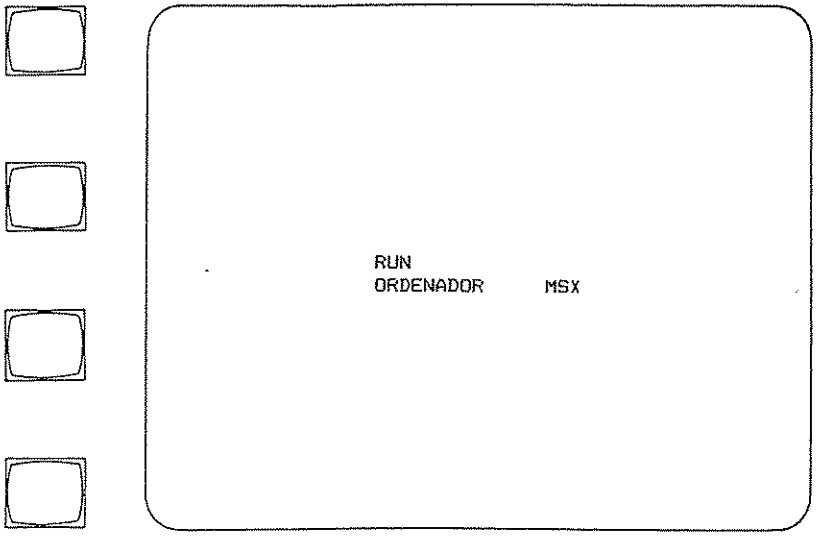
SINTAXIS

SPACE$(<número>)

con un rango de 0 a 255.

EJEMPLO

```
10 A$="ORDENADOR"+SPACE$(5)+"MSX"
20 PRINT A$
```



PUNTOS A RECORDAR

Existe una función similar, SPC, que escribe espacios, pero sólo se puede emplear en instrucciones PRINT y LPRINT.

PRECAUCIONES

Si igualamos el resultado de esta función a una variable numérica se producirá un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

SPC

Parte primera, tema 9: "Algo más sobre las salidas (PRINT) y la pantalla".

SPC eSPaCio

DESCRIPCION

Esta instrucción deja espacios en la pantalla. Sólo se puede emplear dentro de PRINT y LPRINT. Se utiliza principalmente para no gastar tanta memoria cuando necesites incorporar demasiados espacios en una instrucción PRINT.

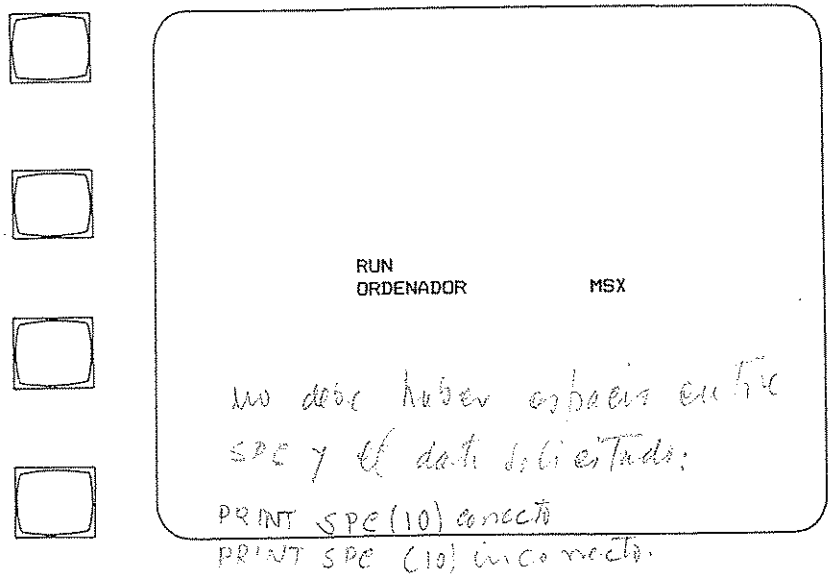
SINTAXIS

SPC(<número>)

rango de 0 a 255.

EJEMPLO

```
10 A$="ORDENADOR":B$="MSX"
20 PRINT A$;SPC(10);B$
```



PUNTOS A RECORDAR

Esta instrucción es diferente a SPACE$, ya que un resultado no se puede tratar como si fuese una cadena de caracteres.

PRECAUCIONES

En caso de que el argumento fuese mayor que 255, se produciría un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT
LPRINT
SPACE$

Parte primera, tema 9: "Algo más sobre las salidas (PRINT) y la pantalla".

SPRITE ON/OFF/STOP

DESCRIPCION

Esta es una de las instrucciones que controlan el manejo de las interrupciones de *sprites*. Activa (ON) o desactiva (OFF/STOP) la detección del choque de dos *sprites* dentro de un programa BASIC.

Se ha de ejecutar SPRITE ON para poder detectar el choque de dos *sprites* y hacer que el ordenador salte a la subrutina apuntada por ON SPRITE GOSUB. Después de ejecutarla, el ordenador analizará si han chocado dos *sprites* antes de atender a la siguiente instrucción. Cuando se haya detectado el choque, se saltará inmediatamente a la subrutina de tratamiento.

Si ejecutas SPRITE STOP el ordenador no tratará los choques que se hayan producido, pero los recordará, y una vez ejecutes SPRITE ON se analizará si se produjo algún choque entre *sprites* y lo tratará si lo hubiere.

SPRITE OFF desactiva la detección de choques entre *sprites*. No se memorizarán aquellos choques que se produzcan mientras dure la ejecución de la instrucción SPRITE OFF.

SINTAXIS

SPRITE ON
SPRITE OFF
SPRITE STOP

EJEMPLO

En este ejemplo verás dos *sprites* cuadrados, uno amarillo y otro blanco, aproximándose desde cada lado de la pantalla. Cuando choquen el BASIC saltará a la subrutina de tratamiento. La orden SPRITE OFF hace que no se vuelva a detectar otro choque. (Prueba este pequeño programa sin SPRITE OFF y observa lo que ocurre.)

```
10  ON SPRITE GOSUB 110
20  SCREEN 2,0
30  SPRITE$(0)=STRING$(8,CHR$(255))
40  SPRITE$(1)=STRING$(8,CHR$(255))
50  SPRITE ON
60  FOR I=10 TO 240
70  PUT SPRITE 0,(I,100),11,0
80  PUT SPRITE 1,(250-I,100),15,1
90  NEXT I
100 END
105 REM Subrutina de choque de sprites
110 SPRITE OFF
120 BEEP
130 RETURN
```


LINEA 10 FIJA LA SUBRUTINA DE TRATAMIENTO EN LA LINEA 110.
LINEA 20 SE PONE EN MODO GRAFICO DE ALTA RESOLUCION.
LINEA 30 EL SPRITE 0 ES UN CUADRADO.
LINEA 40 EL SPRITE 1 TAMBIEN ES OTRO CUADRADO.
LINEA 50 ACTIVA LA DETECCION DEL CHOQUE.
LINEA 60 BUCLE.
LINEA 70 EL SPRITE (AMARILLO) SE ACERCA DESDE LA IZQUIERDA.
LINEA 80 EL SPRITE (BLANCO) SE ACERCA DESDE LA DERECHA.
LINEA 90 NEXT DEL BUCLE.
LINEA 100 FIN.
LINEA 110 IMPIDE LA DETECCION DE MAS CHOQUES.
LINEA 120 SONIDO INDICADOR.
LINEA 130 RETORNA AL BUCLE.

PUNTOS A RECORDAR

Antes de ejecutar SPRITE ON/OFF/STOP se ha de ejecutar previamente ON SPRITE GOSUB; de otro modo el ordenador haría caso omiso y no detectaría los choques.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON SPRITE GOSUB
SPRITE$
PUT SPRITE

Parte segunda, tema 43: "Gráficos avanzados (IV): los *sprites*".

SPRITE$

DESCRIPCION

Es la instrucción con la que se diseñan los *sprites*. Puedes tener hasta 256 en caso de que emplees los tamaños 0 ó 1 (reducidos), o 64 en caso de que los tamaños sean 2 ó 3 (ampliados).

La longitud del campo de diseño del *sprite* es de 32 bytes; en caso de que el *sprite* sea pequeño sólo habrás de declarar los ocho primeros caracteres: el resto se declaran automáticamente, como CHR$(0). Si el *sprite* es de 16 × 16, entonces habrás de declarar los 32 bytes.

En el tema de *sprites* de la *Guía avanzada de programación* encontrarás todo tipo de detalles sobre cómo construir *sprites*.

SINTAXIS

SPRITE$(<entero>)=<car-exp>

EJEMPLOS

```
10 SCREEN 2
20 SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255
)+CHR$(255)+CHR$(112)+CHR$(48)+CHR$(16)
70 PUT SPRITE 0,(100,100),15,0
80 GOTO 80
```

LINEA 10 PANTALLA 2 DE ALTA RESOLUCION.
LINEA 20 DEFINE EL SPRITE 0.
LINEA 70 SITUA EL SPRITE EN EL PUNTO (100,100) CON COLOR BLANCO EN EL PLANO DE SPRITE 0.

El resultado será una flecha en medio de la pantalla.

PUNTOS A RECORDAR

Sólo puedes emplear los *sprites* en los modos 1, 2 y 3.

El tamaño de los *sprites* lo fijas con la instrucción SCREEN; esta instrucción limpia todos los posibles *sprites* que hubiera.

No podrás combinar dos tamaños diferentes de *sprites*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PUT SPRITE
SCREEN
SPRITE ON/OFF/STOP

Parte segunda, tema 43: "Gráficos avanzados (IV): los *sprites*".

SQR Raíz cuadrada *(SQuare Root)*

DESCRIPCION

Esta es la función que calcula la raíz cuadrada.

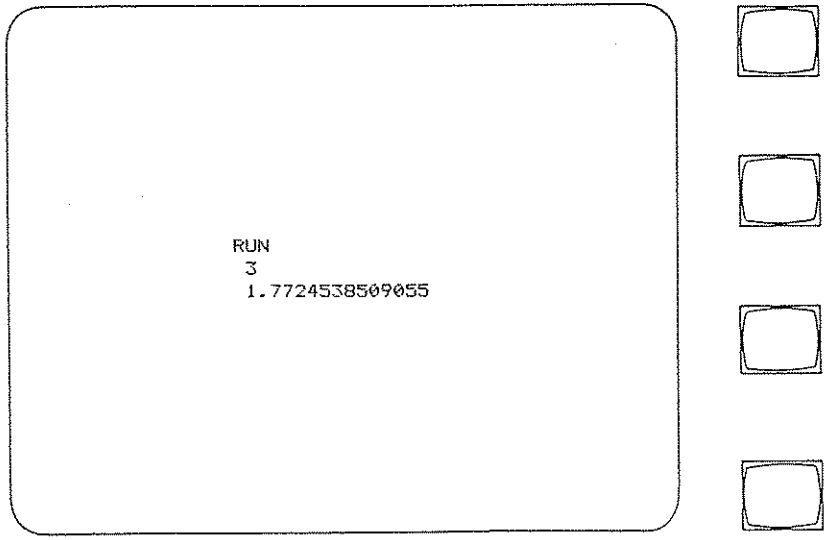
El parámetro pasado debe ser positivo. El MSX no maneja números complejos.

SINTAXIS

SQR(<número>)

EJEMPLO

```
10 PRINT SQR(9)
20 PRINT SQR(ATN(1)*4)
```



PUNTOS A RECORDAR

La función SQR devuelve números de doble precisión.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 19: "Funciones matemáticas".

STEP

DESCRIPCION

STEP es una parte del bucle FOR/TO/NEXT. Esta parte es la que indica el incremento de la variable del bucle en cada iteración.

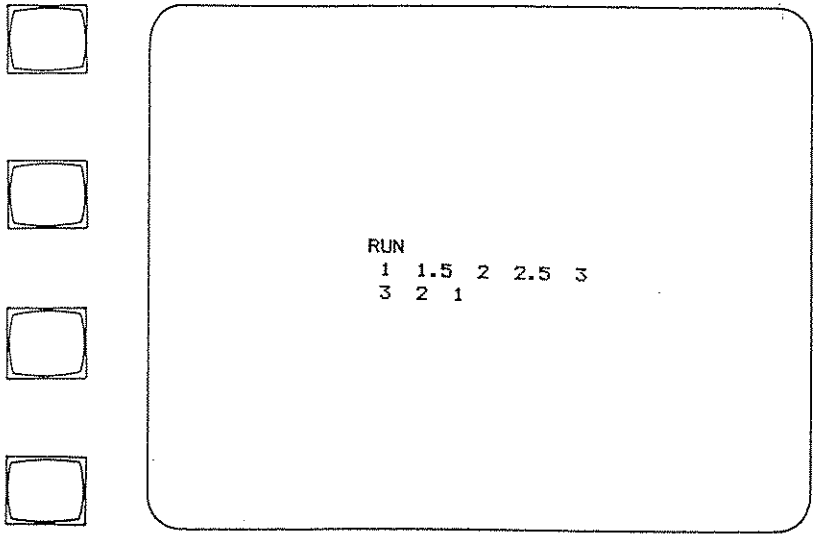
Este incremento puede ser menor que 1 o incluso negativo. Si el incremento es 1 no hace falta indicarlo, ya que la instrucción FOR/TO/NEXT lo hará implícitamente.

SINTAXIS

FOR <num-var> = <número> TO <número> STEP <número>

EJEMPLO

```
10 FOR I=1 TO 3 STEP .5
20 PRINT I;
30 NEXT I
40 PRINT
50 FOR J=3 TO 1 STEP -1
60 PRINT J;
70 NEXT J
```



PRECAUCIONES

FOR I = 1 TO 10 STEP −1

Este es un error bastante frecuente, sólo se repetirá una vez.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

FOR
TO
NEXT

Parte primera, tema 5: "Cómo emplear los bucles".

STICK *joySTICK*

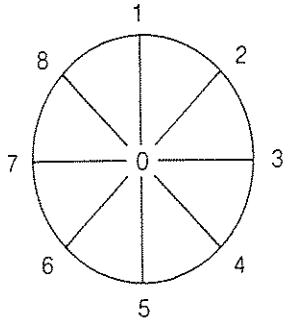
DESCRIPCION

Esta función devuelve la dirección que apunta el *joystick* o las teclas del cursor, en caso de que las emplees como *joystick*.

STICK (<n>)... <n> puede ser 0, 1 ó 2.

n = 0 teclas del cursor.
n = 1 *joystick* 1.
n = 2 *joystick* 2.

Esta función te devuelve el valor 0 en caso de que el *joystick* no apunte a ningún lado. Los valores que se devuelven según la dirección apuntada son



Si empleas las teclas del cursor habrás de pulsar dos de ellas a la vez para moverte diagonalmente; por ejemplo, pulsa ↑ y → para tomar la dirección 2.

SINTAXIS

STICK(<n>)

EJEMPLO

En este ejemplo mostramos los valores de las direcciones, empleando las teclas del cursor como *joystick*.

```
10 A=STICK(0)
20 LOCATE 15,10 : PRINT A
30 GOTO 10
```


PUNTOS A RECORDAR

- Esta función siempre te devolverá un valor entero entre 0 y 8.
  Para saber el estado del disparador del *joystick* emplea la función STRIG.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

STRIG

Parte segunda, tema 51: "Periféricos".

STOP

DESCRIPCION

Esta instrucción detiene la ejecución de un programa BASIC y devuelve el control al modo directo.

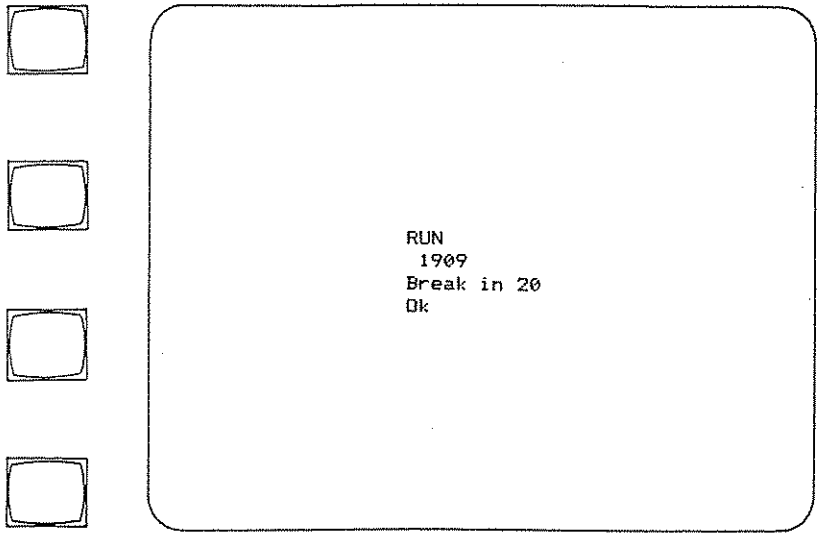
Después de ejecutarse esta instrucción se visualizará el mensaje "*Break in <línea>*" ("Parado en la <línea>").

SINTAXIS

STOP

EJEMPLO

```
10 PRINT 1909
20 STOP
```



PUNTOS A RECORDAR

La instrucción STOP no cierra (CLOSE) ningún fichero al finalizar el programa, mientras que END sí lo hace.

Puedes continuar la ejecución desde la siguiente línea al comando STOP mediante CONT.

Emplea tantas órdenes STOP como sea necesario dentro de tu programa.

Esta instrucción es totalmente diferente a STOP/ON/OFF/STOP.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

END

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

STOP ON/OFF/STOP

DESCRIPCION

Esta instrucción es bastante diferente de STOP, la cual detiene la ejecución de un programa. STOP ON/OFF/STOP activa y desactiva la detección de las teclas <CTRL><STOP>. Al ejecutar STOP ON el BASIC analiza si se ha pulsado <CTRL><STOP> antes de ejecutar cada instrucción. Sí así fuese, entonces el BASIC saltaría a la subrutina apuntada por ON STOP GOSUB, ejecutada previamente a STOP ON.

Si ejecutas STOP STOP entonces el ordenador no tratará las interrupciones de <CTRL><STOP>, pero recordará si hubo alguna de ellas y pasará a tratarla una vez hagas STOP ON.

En cambio STOP OFF desactivará totalmente la detección de cualquier interrupción de este tipo.

SINTAXIS

STOP ON
STOP OFF
STOP STOP

EJEMPLO

Este programa te muestra cómo ON STOP GOSUB evita que nadie detenga la ejecución del programa. La línea 20 activa la detección. Si pulsas <CTRL><STOP> te mostrará <detección desactivada> y continuará ejecutando el programa. Este programa tiene una subrutina especial de salida. Pulsa "s" para activar las teclas <CTRL><STOP>; de otro modo te escribirá MSX indefinidamente.

```
10  ON STOP GOSUB 100
20  STOP ON
30  IF INKEY$="s" THEN STOP OFF : PRINT "<CTRL><ST
OP> activado"
40  PRINT "MSX"
50  GOTO 30
100 BEEP
110 PRINT "<CTRL><STOP> desactivado"
120 RETURN
```

LINEA 10 APUNTA A LA SUBRUTINA DE TRATAMIENTO.
LINEA 20 ACTIVA LA DETECCION DE <CTRL><STOP>.
LINEA 30 DESACTIVA ESTA DETECCION Y TE ESCRIBE UN MENSAJE.
LINEA 40 ESCRIBE MSX.

LINEA 50 VUELVE A LA LINEA 30.
LINEA 100 SONIDO (BEEP) INDICADO.

LINEA 110 MENSAJE.
LINEA 120 VUELVE AL PUNTO DESDE DONDE SE DETECTO <CTRL><STOP>.

PUNTOS A RECORDAR

Observa que ON STOP no evita la parada mediante la tecla <STOP>. Solamente evita que rompas la ejecución del programa. Intenta pulsar la tecla <STOP> en el programa anterior: observarás que se detiene y aparece el cursor en pantalla; si pulsas de nuevo STOP se continuará con la ejecución.

No olvides STOP ON para poder detectar <CTRL><STOP>.

Este tipo de interrupciones están desactivadas si el programa no se está ejecutando o está dentro de las subrutinas de tratamiento de errores.

PRECAUCIONES

Para emplear esta instrucción has de tener una orden ON STOP GOSUB y una subrutina de tratamiento.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON STOP GOSUB

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

STR$

DESCRIPCION

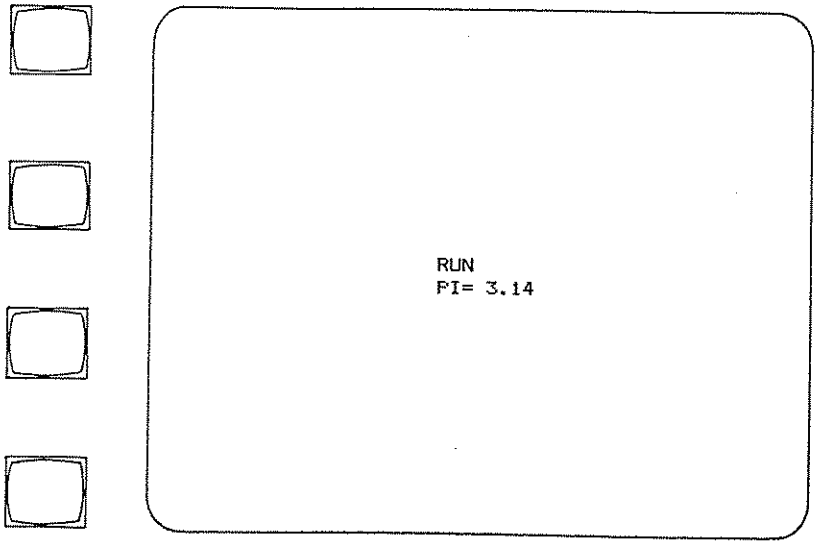
STR$ convierte un argumento numérico a cadena de caracteres.

SINTAXIS

STR$ (<número>)

EJEMPLO

```
10 A$="PI="+STR$(3.14)
20 PRINT A$
```



PUNTOS A RECORDAR

La función inversa de ésta es VAL.

PRECAUCIONES

El argumento ha de ser siempre un número.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

VAL

Parte primera, tema 15: "Funciones".
Parte segunda, tema 31: "Conversión de tipos".

STRIG

DESCRIPCION

Esta función devuelve el estado del disparador de un determinado *joystick*.

SINTAXIS

STRIG (<n>)

<n>=0	Barra espaciadora (empleada como disparador).
<n>=1,3	*Joystick* 1.
<n>=2,4	*Joystick* 2.

STRIG(<n>)= 0	sin pulsar.
STRIG(<n>)= -1	pulsando.

EJEMPLO

Si pulsas la barra espaciadora tendrás un mensaje.

```
10 IF STRIG(0) THEN PRINT "PULSASTE LA BARRA ESPAC
IADORA"
20 GOTO 10
```

PUNTOS A RECORDAR

Esta función se suele emplear en juegos de tiro al blanco.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

STICK

Parte segunda, tema 51: "Periféricos".

STRIG ON/OFF/STOP

DESCRIPCION

Esta instrucción es totalmente diferente de STRIG, que devuelve el estado de los disparadores. STRIG(<n>) ON/OFF/STOP activa/desactiva la detección de las interrupciones de los disparadores. Al ejecutar STRIG(<n>) ON, el BASIC analiza si se ha producido una interrupción por parte de ese disparador antes de ejecutar cualquier instrucción. Si así fuese saltaría a la subrutina específica apuntada por la instrucción ON STRIG GOSUB, que debe ejecutarse antes de STRIG(<n>) ON.

Si ejecutas STRIG(<n>) STOP el BASIC no tratará las interrupciones del disparador <n>, pero las recordará y al encontrar STRIG(<n>) ON las tratará si las hubiera habido.

En cambio STRIG(<n>) OFF desactivará totalmente la detección de interrupciones del disparador <n>.

SINTAXIS

STRIG(<n>) ON
STRIG(<n>) STOP
STRIG(<n>) OFF

siendo <n> el número del disparador.

Hay cinco disparadores, y éstos son:

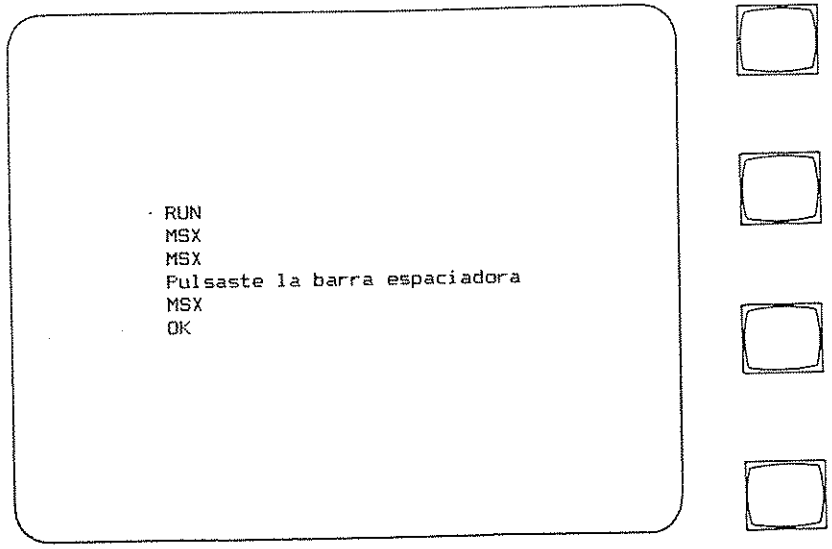
0 = barra espaciadora
1 = disparador 1 del *joystick* 1.
2 = disparador 1 del *joystick* 2.
3 = disparador 2 del *joystick* 1.
4 = disparador 2 del *joystick* 2.

EJEMPLO

Aquí tienes un ejemplo en el que te mostramos cómo funcionan las subrutinas de detección de interrupciones de los disparadores (en este caso empleamos la barra espaciadora como disparador). Cuando pulses la barra espaciadora saltará a la subrutina de tratamiento; de otro modo escribiría infinitamente "MSX" hasta que pulses la tecla <s>.

```
10  ON STRIG GOSUB 100
20  STRIG(0) ON
30  IF INKEY$="s" THEN END
40  PRINT "MSX"
50  GOTO 30
90  REM Rutina de la barra espaciadora
100 BEEP
110 PRINT "Pulsaste la barra espaciadora"
120 RETURN
```


LINEA 10 FIJA LA SUBRUTINA DE TRATAMIENTO EN LA LINEA 100.
LINEA 20 ACTIVA LA DETECCION DE LA INTERRUPCION.
LINEA 30 SI PULSAS <s> TERMINA.
LINEA 40 ESCRIBE MSX.
LINEA 50 VUELVE A LA LINEA 30.
LINEA 100 SONIDO (BEEP) INDICADOR.
LINEA 110 MENSAJE.
LINEA 120 VUELVE AL PUNTO DONDE DETECTO LA INTERRUPCION.



PUNTOS A RECORDAR

Las interrupciones de este tipo no funcionan si el programa no se está ejecutando o dentro de las subrutinas de tratamiento de errores.

PRECAUCIONES

Un error muy común es escribir: STRIG (0) ON; esto no es correcto, ya que no debe haber ningún espacio entre STRIG y (0).

Si en la instrucción ON STRIG GOSUB apuntas a una línea de comienzo de subrutina inexistente, tendrás un error de línea no definida *(Undefined line number)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ON STRIG GOSUB
STRIG

Parte segunda, tema 37: "Sucesos e interrupciones en BASIC".

STRING$

DESCRIPCION

Esta función te devuelve una cadena con un cierto número de caracteres especificados por ti.

SINTAXIS

STRING$(<n>,<código ASCII>)

con lo que se devuelve una cadena de *n* caracteres, siendo éstos los correspondientes al código ASCII dado.

STRING$(<longitud>,<cadena de caracteres>)

que te devuelve una cadena de la longitud especificada y con el primer carácter de la cadena dada.

<n>,<longitud> pueden ser variables o constantes.

EJEMPLO

```
10 INPUT A
20 PRINT STRING$(A,"X")
30 GOTO 10
```

```
RUN
 10
XXXXXXXXXX
 30
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

PUNTOS A RECORDAR

Esta función es muy útil para definir un *sprite* cuadrado.

SPRITE$(0)=STRING$(8,CHR$(255))

PRECAUCIONES

Si la longitud de la cadena excede de la máxima longitud permitida a cadenas de caracteres tendrás un error de exceso de caracteres *(Out of string space);* esta longitud es de 200 de modo estándar, pero puedes cambiarla con CLEAR.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 20: "El código ASCII".

SWAP

DESCRIPCION

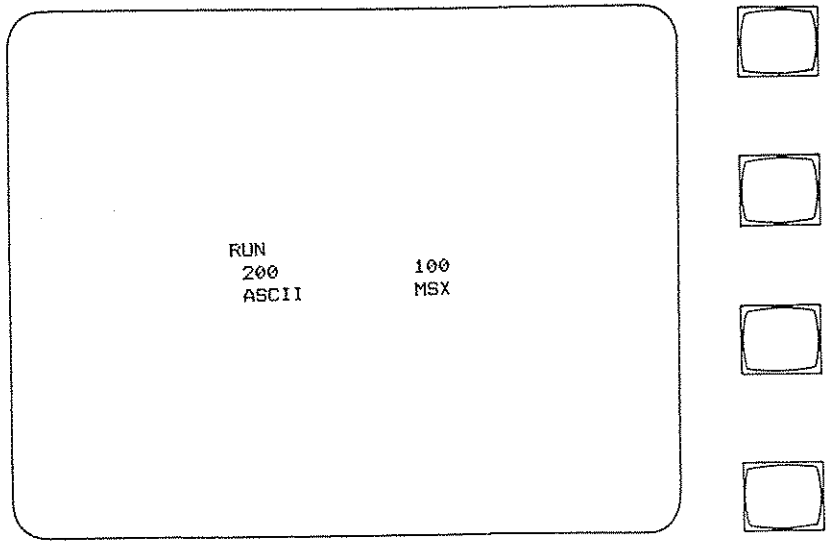
Estas instrucciones intercambian el contenido de dos variables.

SINTAXIS

SWAP<variable>,<variable>

EJEMPLO

```
10 A=100 : B=200
20 C$="MSX" : D$="ASCII"
30 SWAP A,B
40 SWAP C$,D$
50 PRINT A,B
60 PRINT C$,D$
```



PRECAUCIONES

Si intentas intercambiar los contenidos de variables de distinto tipo se producirá un error en el tipo de datos *(Type mismatch).*

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 6: "Las condiciones".

TAB

DESCRIPCION

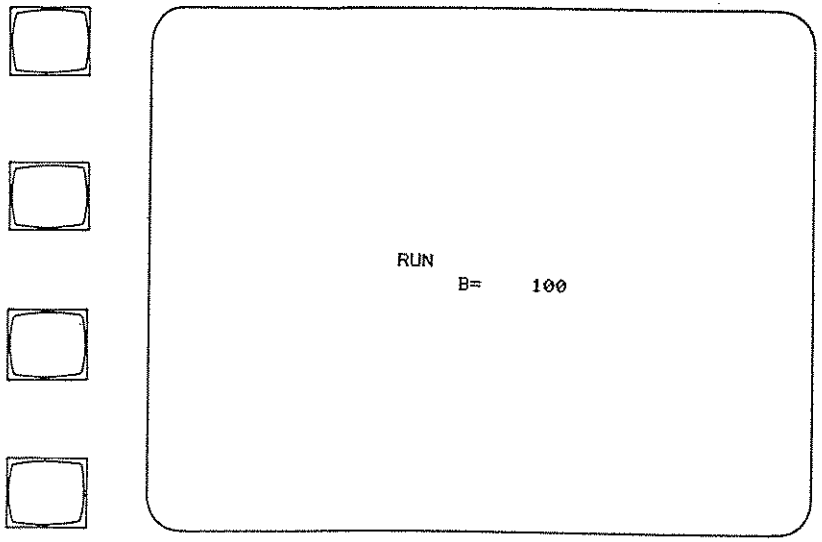
Esta instrucción se emplea siempre en combinación de PRINT y LPRINT; tabula dejando un número determinado de espacios a la izquierda.

SINTAXIS

PRINT TAB(<número>)
LPRINT TAB(<número>)

EJEMPLO

```
10 B=100
20 PRINT TAB(5)"B=" TAB(10) B
```



PUNTOS A RECORDAR

<número> ha de ser menor que WIDTH −1 o saltará a la siguiente línea.
En caso de que el número fuese real se truncaría su parte decimal.
La instrucción LOCATE es mucho más flexible que ésta, ya que puede especificar el punto donde se sitúa el cursor.

PRECAUCIONES

Si el argumento es mayor que 255 tendrá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

PRINT
LPRINT
LOCATE

Parte primera, tema 9: "Algo más sobre las salidas (PRINT) y la pantalla".

TAN TANgente

DESCRIPCION

Esta función te devuelve la tangente del ángulo especificado en radianes. El resultado devuelto es de doble precisión.

SINTAXIS

TAN(<número>)

EJEMPLO

```
PRINT TAN(0.5)
.54630248984381
```

PRECAUCIONES

Tanto el argumento como el resultado son de tipo numérico. Si los tomas como si fueran cadenas de caracteres tendrás un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

ATN
COS
SIN

Parte primera, tema 19: "Funciones matemáticas".

THEN

DESCRIPCION

Esto es parte de la instrucción compuesta IF/THEN/ELSE. Si la condición se cumple, entonces se ejecutan las instrucciones que siguen a THEN.

Si hubiese un número de línea después de THEN, se tomará como la abreviación de GOTO <línea>.

SINTAXIS

IF <condición> THEN <instrucciones>
IF <condición> THEN <instrucciones> ELSE <instrucciones>
IF <condición> THEN <línea>

EJEMPLO

```
10 INPUT X
20 IF X=10 THEN PRINT "SI"
30 GOTO 10
```

```
RUN
? 12
? 10
SI
```

PUNTOS A RECORDAR

La palabra THEN se puede sustituir por GOTO en el caso de que saltes a otra línea.

IF X = 1 GOTO 2000

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

IF
GOTO
ELSE

Parte primera, tema 6: "Las condiciones".
Parte segunda, tema 35: "Algebra de Boole (II): la sentencia IF/THEN/ELSE".

TIME

DESCRIPCION

Esta función te devuelve el valor de reloj interno al sistema.

Al conectar el ordenador se inicializa a 0 y se incrementa cada vez que el procesador de la pantalla de video (VDP) hace una interrupción; esto sucede cincuenta veces por segundo.

Puedes darle cualquier valor al reloj del sistema mediante TIME. Puedes incluso ponerlo de nuevo a 0.

SINTAXIS

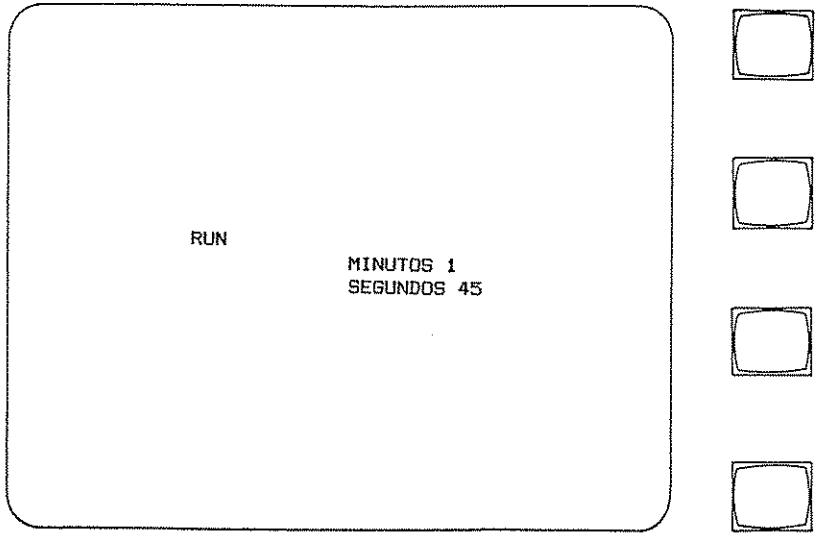
TIME

EJEMPLO

Reloj digital.

```
10 CLS
20 TIME=0
30 MIN%=TIME/3600
40 SEG%=TIME/60-MIN%*60
50 LOCATE 10,11 : PRINT "MINUTOS";MIN%
60 LOCATE 10,12 : PRINT "SEGUNDOS";SEG%
70 GOTO 30
```

RESULTADO: Verás un reloj digital en el centro de la pantalla.



PUNTOS A RECORDAR

Cuando se desactiven las interrupciones del VDP el reloj se parará. Esto sucederá cuando estés utilizando algún periférico como el cassette al cargar un programa.

El valor que devuelve esta función siempre es un entero positivo.

También se emplea esta función para obtener números auténticamente aleatorios mediante RND(-TIME).

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

RND

Parte primera, tema 15: "Funciones".

TO

DESCRIPCION

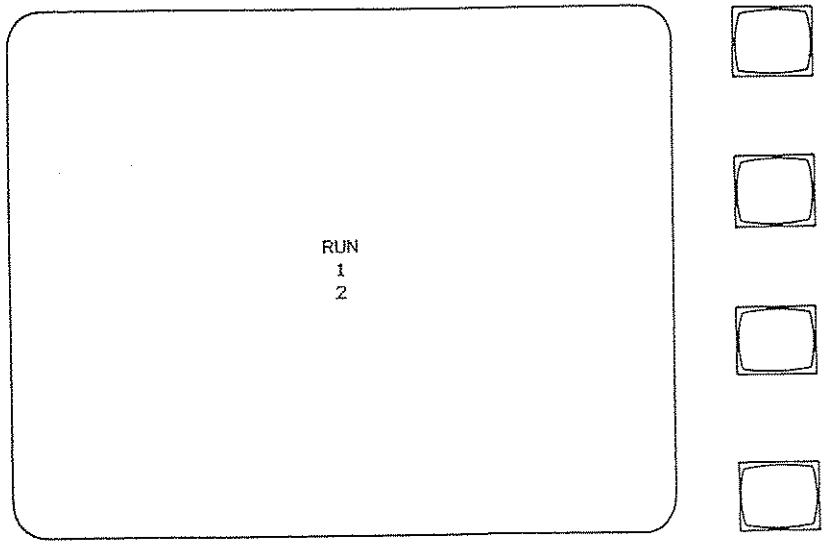
Es una parte de la estructura FOR ... TO ... STEP ... NEXT, que siempre debe ir después de FOR. El valor que precede a TO debe iniciar el bucle y el que le sigue lo finaliza.

SINTAXIS

FOR <var-num> = <número> TO <número>
FOR <var-num> = <número> TO <número> STEP <número>

EJEMPLO

```
10 FOR I=1 TO 2
20 PRINT I
30 NEXT I
```



PUNTOS A RECORDAR

No necesitas dejar ningún espacio entre los valores inicial y final del bucle y la palabra TO, pero así será más fácil de leer.

FOR I = 1TO100

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

FOR
STEP
NEXT

Parte primera, tema 5: "Cómo emplear los bucles".

TROFF

DESCRIPCION

Esta instrucción desactiva la visualización de la traza de un programa durante su ejecución. Desactiva la orden TRON (es la única forma de desactivarla).

SINTAXIS

TROFF

PUNTOS A RECORDAR

Para poder ver la traza de un programa emplea TRON.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

TRON

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

TRON

DESCRIPCION

Esta instrucción hace que el ordenador te muestre por la pantalla el número de línea que está ejecutando.

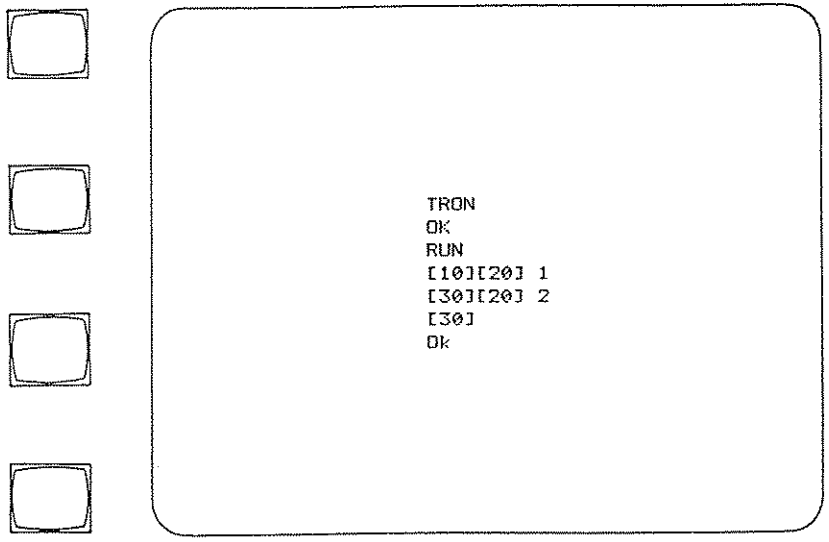
Sólo se emplea para depurar programas, siempre que sean liosos de seguir de otro modo.

SINTAXIS

TRON

EJEMPLO

```
10 FOR I=1 TO 2
20 PRINT I
30 NEXT I
```



PUNTOS A RECORDAR

Si son demasiados los números de línea a escribir no conseguirás entender nada. No funciona en pantallas gráficas.

Para desactivar la orden TRON emplea la instrucción TROFF.

PRECAUCIONES

Se emplea para depurar programas.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

TROFF

Parte primera, tema 7: "Comandos útiles e indicaciones para escribir programas".

USR

DESCRIPCION

Esta función llama a una subrutina en código máquina definida por el usuario desde el BASIC. La dirección donde comienza esta subrutina se declara con la instrucción DEFUSR.

Puedes pasar un parámetro a esta función, así como ésta puede pasarte un resultado.

SINTAXIS

USR[<dígito>](<argumento>)

<dígito> = 0...9

<argumento> (de cualquier tipo) parámetro a pasar a la subrutina en código máquina.

EJEMPLO

```
PRINT USR0("QWERTY")
DMY=USR4(B%)
X=USR9(1000)
```

PUNTOS A RECORDAR

En esta parte no se explica cómo emplear el código máquina. Si deseas más detalles consulta el capítulo que trata sobre la función USR.

PRECAUCIONES

Cuando emplees subrutinas en código máquina creadas por ti hazlo con cuidado. Graba (CSAVE) tu programa antes de ejecutarlo, ya que si tienes algún fallo en dicha subrutina te será prácticamente imposible recuperar el programa.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

DEFUSR

Parte segunda, tema 49: "Función USR y el código máquina".

VAL VALor

DESCRIPCION

Esta función devuelve el valor numérico de una cadena de caracteres, todos ellos numéricos. Si la cadena es la representación de un número se le podrá aplicar esta función. Sin embargo, no tendrá validez sobre cadenas que contengan representaciones de expresiones aritméticas.

La función VAL hará omisión de los espacios, tabulaciones y saltos de línea que haya delante de un número (véanse ejemplos). Sin embargo, no dará el valor pedido si fueran otros caracteres diferentes delante del número; por ejemplo, VAL ("ZWY 100") no dará 100 como resultado.

Esta función reconoce los signos positivo (+) y negativo (−).

SINTAXIS

VAL(<cadena de caracteres>)

EJEMPLO

```
PRINT VAL("          -100")
-100
PRINT VAL("CABALLO 1000")
0
A$="+10":PRINT VAL(A$)
10
```

PUNTOS A RECORDAR

La función inversa de VAL es STR$.

PRECAUCIONES

VAL("−100 + 900") devolverá −100; esta función no trabaja con las expresiones dentro de una cadena.

VAL(A%) dará un error en el tipo de datos *(Type mismatch)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

STR$

Parte primera, tema 15: "Funciones".
Parte segunda, tema 31: "Conversión de tipos".

VARPTR PunTeRo de VARiable

DESCRIPCION

VARPTR (<nombre de variable>) nos devuelve la dirección del primer byte de memoria asignado al contenido de dicha variable. La variable puede ser de cualquier tipo, pero ha de existir antes de que le apliques esta función.

También te devuelve la dirección de comienzo de un bloque de control de fichero.

La dirección devuelta será un número entero entre −32768 y 32767. Si el número es negativo súmale 65536 para obtener la dirección real.

SINTAXIS

VARPTR (<nombre de variable>)
VARPTR (#<nombre fichero>)

EJEMPLO

```
PRINT VARPTR (A(0))
S$="IIII":PRINT 65536+VARPTR(S$)
D%=100:PRINT "H";HEX$(65536+VARPTR(D%))
```

PUNTOS A RECORDAR

Esta función se emplea para conocer la dirección de comienzo de una matriz y así podérselo pasar a una subrutina en código máquina, haciendo VARPTR (A(0)).

PRECAUCIONES

Si la variable que pasas como parámetro no existe tendrás un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 48: "Mapa de memoria".

VDP Registros del procesador de pantalla *(Video Display Processor register)*

DESCRIPCION

Se emplea para acceder al procesador de pantalla.
Hay nueve registros:

0 ... 7 son para escribir exclusivamente; puedes darles un valor.
8 sólo es para lectura.

Se hablará más detalladamente de estos registros en el tema "Parte gráfica: acceso al procesador de video (VDP)".

SINTAXIS

Para los registros del 0 al 7, <dígito> = 0 ... 7

VDP(<dígito>)=<número>

Registro 8

<var-num> = VDP(8)

EJEMPLO

```
PRINT "REGISTRO DE ESTADO DEL VDP"; BIN$(VDP(8)).
```

Este ejemplo te muestra el contenido del registro de estado del VDP con representación binaria.

PUNTOS A RECORDAR

El VDP de la versión inglesa del MSX es el TMS9929A, compatible con el sistema de televisión PAL.

PRECAUCIONES

Si desconoces la función del VDP posiblemente estropees lo que hay en pantalla. Si se descontrola la pantalla, apaga el ordenador y enciéndelo de nuevo.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte segunda, tema 44: "Gráficos avanzados (V): acceso al procesador de video (VDP)".

VPEEK Video RAM PEEK

DESCRIPCION

Te devuelve el contenido de una posición de la RAM de video.

SINTAXIS

VPEEK(<dirección>)
<dirección> = 0 ... 16383

EJEMPLO

```
PRINT VPEEK(100)
V%=VPEEK(999)
```

PUNTOS A RECORDAR

Te recomendamos que emplees esta función si eres un programador experto.

Al estar la RAM de video separada de la memoria principal se necesita de esta función.

Todos los MSX tienen 16K bytes de RAM de video.

Esta función es la inversa de VPOKE.

PRECAUCIONES

Si la <dirección> no está dentro del rango indicado, entonces habrá un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

VPOKE

Parte segunda, tema 45: "Gráficos avanzados (VI): la RAM de video".

VPOKE Video RAM POKE

DESCRIPCION

Le da un determinado valor a un byte de la RAM de video. Debes indicar la dirección y el valor a dar al byte.

El valor del byte ha de estar entre 0 y 255.

SINTAXIS

VPOKE <dirección>,<byte>
<dirección> 0 ... 16383
<byte> 0 ... 255

EJEMPLO

```
VPOKE 998,255
```

PUNTOS A RECORDAR

No emplees esta orden a no ser que domines lo que estás haciendo.

Esta función existe debido a que la RAM de video y la memoria principal están separadas.

Todos los ordenadores MSX tienen 16K bytes de RAM de video.

Esta función es la inversa de VPEEK.

PRECAUCIONES

Si la <dirección> se sale de su rango correspondiente tendrás un error de desbordamiento *(Overflow error)*; en cambio, si es el valor del <byte> el error será de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

VPEEK

Parte segunda, tema 45: "Gráficos avanzados (VI): la RAM de video".

WAIT

DESCRIPCION

Esta instrucción detiene la ejecución y espera a que un determinado puerto de entrada tome un valor binario específico.

SINTAXIS

WAIT <número puerto>,<I>[,<J>]

Rangos:

<número puerto> = 0 ... 255
<I> y <J> = 0 ... 255

PUNTOS A RECORDAR

Con el valor leído en el puerto se hace la OR-exclusiva (XOR) con J, y con el resultado se hace el Y lógico (AND) con I. Si el resultado de todo esto es 0 se vuelve a leer el valor del puerto y operarlo de nuevo. Si se omite el valor de J se le asigna 0 automáticamente.

Si el resultado es distinto de 0 se continúa la ejecución.

Esta instrucción accede directamente al puerto, sin emplear las BIOS. Por tanto, es muy posible que si utilizas esta instrucción el programa pierda compatibilidad con los MSX para hacerse dependiente del código máquina.

El MSX sólo trabaja con los puertos 0 a 255; cualquiera fuera de este rango será omitido.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

OUT
INP

WIDTH

DESCRIPCION

Esta orden fija la anchura de la pantalla; también la borra al ejecutarse.

modo de pantalla	estándar	máximo
0	37*	40
1	29	32

SINTAXIS

WIDTH <número>

EJEMPLO

WIDTH 30
WIDTH 20

PUNTOS A RECORDAR

La anchura mínima es de un carácter.

PRECAUCIONES

Si el argumento se sale del rango correspondiente tendrás un error de llamada ilegal a una función *(Illegal function call)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

Parte primera, tema 21: "Modos de pantalla".

* 39 en la versión americana y europea.

XOR OR-eXclusivo

DESCRIPCION

Es uno de los operadores lógicos que tienen el álgebra de Boole:

XOR	X	Y	X XOR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	0

luego 100 XOR 50 se calculará:

	X	100	0000000001100100
XOR	Y	50	0000000000110010
		86	0000000001010110

SINTAXIS

<var-num> = <número> XOR <número>

PUNTOS A RECORDAR

Este operador verifica la siguiente relación:

A = A XOR Y XOR Y

PRECAUCIONES

Si algún operando se sale del rango de los enteros habrá un error de desbordamiento *(Overflow error)*.

INSTRUCCIONES RELACIONADAS Y REFERENCIAS

AND, OR, EQV, NOT, IMP

Parte segunda, tema 34: "Algebra de Boole" (I).

PARTE CUARTA

EL SISTEMA OPERATIVO

Esta sección trata de las BIOS *(Basic Input Output System*, rutinas de entrada y salida), explica el modo de emplear los puntos de entrada y una tabla de la RAM del sistema. Para entender esta parte has de tener un cierto conocimiento del lenguaje ensamblador del Z80, que por su extensión no te explicaremos en este libro.

De los temas 53 al 59 se describen las llamadas a BIOS disponibles en cualquier ordenador MSX. Las llamadas relacionadas están agrupadas por temas. El formato que tienen estas descripciones es igual en todas. La primera línea te indica el nombre y la dirección del punto de entrada en hexadecimal de cada rutina. En la siguiente línea se te indica la instrucción del Z80 que has de ejecutar para llamar a la subrutina. Le sigue una explicación de lo que hace, y los parámetros que puedes pasar y tomar de la rutina. Se enumeran, también, los registros y posiciones de memoria que se modifican con la llamada. Finalmente, se te indicará el estado de las interrupciones y posibles llamadas útiles a dicha rutina.

En el tema 60 se explican los vectores, insertándose a continuación una tabla con todos ellos, su dirección y desde dónde se les llama.

El último tema es una tabla de las posiciones de la RAM del sistema, con su dirección y la longitud que tienen (en bytes).

53
Instrucciones RST

Este tema trata las ocho instrucciones que se pueden ejecutar mediante las instrucciones RST del Z80.

RST0	CHKRAM
RST1	SYNCHR
RST2	CHRGTR
RST3	OUTDO
RST4	DCOMPR
RST5	GETYPR
RST6	CALLF
RST7	KEYINT

CHKRAM 0000

Ejecución a través de RST 00.

Es desde donde se comienza la ejecución al conectar tu ordenador.

Llamando a esta subrutina ocasionarás una inicialización total (RESET). El nombre viene del término inglés *check RAM* (comprobación de la RAM), y no explica la totalidad de lo que hace esta rutina.

PARAMETROS DE ENTRADA

No tiene

PARAMETROS DE SALIDA

Esta rutina no devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Todos los registros y cualquier dato que se halle en memoria se pierde.

POSIBLES USOS

Sólo hay una función para esta rutina: inicializar el ordenador (RESET).

SYNCHR 0008

Ejecución a través de RST 1.

Esta rutina se emplea para analizar los errores sintácticos, desde el intérprete de BASIC. Si no hay ninguno el proceso continúa con la llamada a CHRGTR (0010); en caso contrario se dará el mensaje de error sintáctico *(Syntax error)*.

PARAMETROS DE ENTRADA

HL apunta al siguiente carácter del texto BASIC, y el byte que sigue a la instrucción RST1 es el carácter con el que debe ser comparado.

PARAMETROS DE SALIDA

Al final tendrás que HL apunta al siguiente carácter, y que éste está en el registro 1. El *flag* de acarreo estará a 1 si es un número, y el *flag* de cero estará también a 1 si se acabó la instrucción.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

El acumulador, los *flags* y el par de registros HL.

Nota: Esta instrucción no está hecha para los programadores en código máquina; por tanto, es mejor dejarla a un lado.

CHRGTR 0010

Ejecución a través de RST2.

El intérprete de BASIC emplea esta instrucción para tomar el siguiente carácter o *token* del programa. Generalmente se llama desde la rutina SYNCHR (RST1).

PARAMETROS DE ENTRADA

HL debe apuntar al siguiente carácter a traer.

PARAMETROS DE SALIDA

HL apuntará al siguiente carácter a leer, y los *flags* de acarreo y de cero estarán a 1 si se ha leído un número y si se ha llegado al fin de la instrucción actual, respectivamente.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

A los pares de registros AF y HL .

Nota: Al igual que SYNCHR (RST1), es mejor dejar aparte esta rutina.

OUTDO 0018

Ejecución a través de RST3.

Esta instrucción escribe el contenido del acumulador en la unidad seleccionada en dicho momento.

PARAMETROS DE ENTRADA

El acumulador ha de contener el código a escribir. Si dicho código se ha de escribir en un fichero de disco PTRFIL debe contener el puntero a dicho fichero. Si el fichero está en la impresora PTRFIL ha de contener un cero y PRYFLG debe ser distinto de cero. Si el fichero está en un terminal y PTRFIL y PTRFLG han de contener ambas un cero.

PARAMETROS DE SALIDA

Esta rutina no devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Tampoco afecta a ningún registro ni posición de memoria.

Nota: Después de apilar (almacenar en pila) el contenido de AF llama al vector H. OUTD, antes de reanudar la ejecución.

DCOMPR 0020

Ejecución a través de RST4

Esta llamada compara los contenidos de los pares de registros HL y DE del Z80.

PARAMETROS DE ENTRADA

No se necesita ningún parámetro de entrada, aparte de los contenidos de HL y DE.

PARAMETROS DE SALIDA

A la salida tendrás un 1 en el *flag* de acarreo, si HL es menor que DE; en caso contrario habrá un 0. Si HL = DE el *flag* de cero estará a 1.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo se alteran los contenidos del acumulador y los *flags*.

GETYPR 0028

Ejecución a través del RST5.

El intérprete de BASIC emplea esta instrucción para saber qué tipo de acumulador de coma flotante se está empleando.

PARAMETROS DE ENTRADA

No necesita de parámetros de entrada.

PARAMETROS DE SALIDA

Se pondrán algunos *flags* a 1, según el tipo del acumulador de coma flotante que se esté empleando.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina sólo modifica los *flags*.

CALLF 0030

Ejecución a través de RST6.

RST6 hace una llamada a un cartucho interno.

PARAMETROS DE SALIDA

Depende de la rutina llamada.

PARAMETROS DE ENTRADA

RST6 ha de ir seguida del byte descriptor del cartucho; los dos siguientes bytes contienen la dirección de llamada.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

También dependen de la rutina de llamada.

Nota: El byte descriptor de cartucho tiene el siguiente formato:

N.º de bit	7	6	5	4	3	2	1	0
Contenido	F	x	x	x	S	S	P	P

PP	(0-3)	es el número del cartucho primario a seleccionar.
SS	(0-3)	es el número del cartucho secundario a seleccionar.
f	(0-1)	es un *flag;* estará a 1 si se usa el cartucho secundario, o a 0 si no se emplea.

Los bit 4, 5 y 6 no se emplean; su valor es inmutable.

KEYINT 0038

Ejecución a través de RST7 o una interrupción.

El MSX trabaja con el modo 1 de interrupción, o sea, es la instrucción a ejecutar cuando se produce una interrupción enmascarable. Las interrupciones se producen cada 8,02 segundos por el reloj (50 Hz). Cuando se produce ésta se decrementa el intervalo, JIFFY se incrementa, se actualizan las escalas musicales y se analizan los disparadores del *joystick* y el teclado.

PARAMETROS DE ENTRADA

Los pasados por el *hardware* puede no tenerlos.

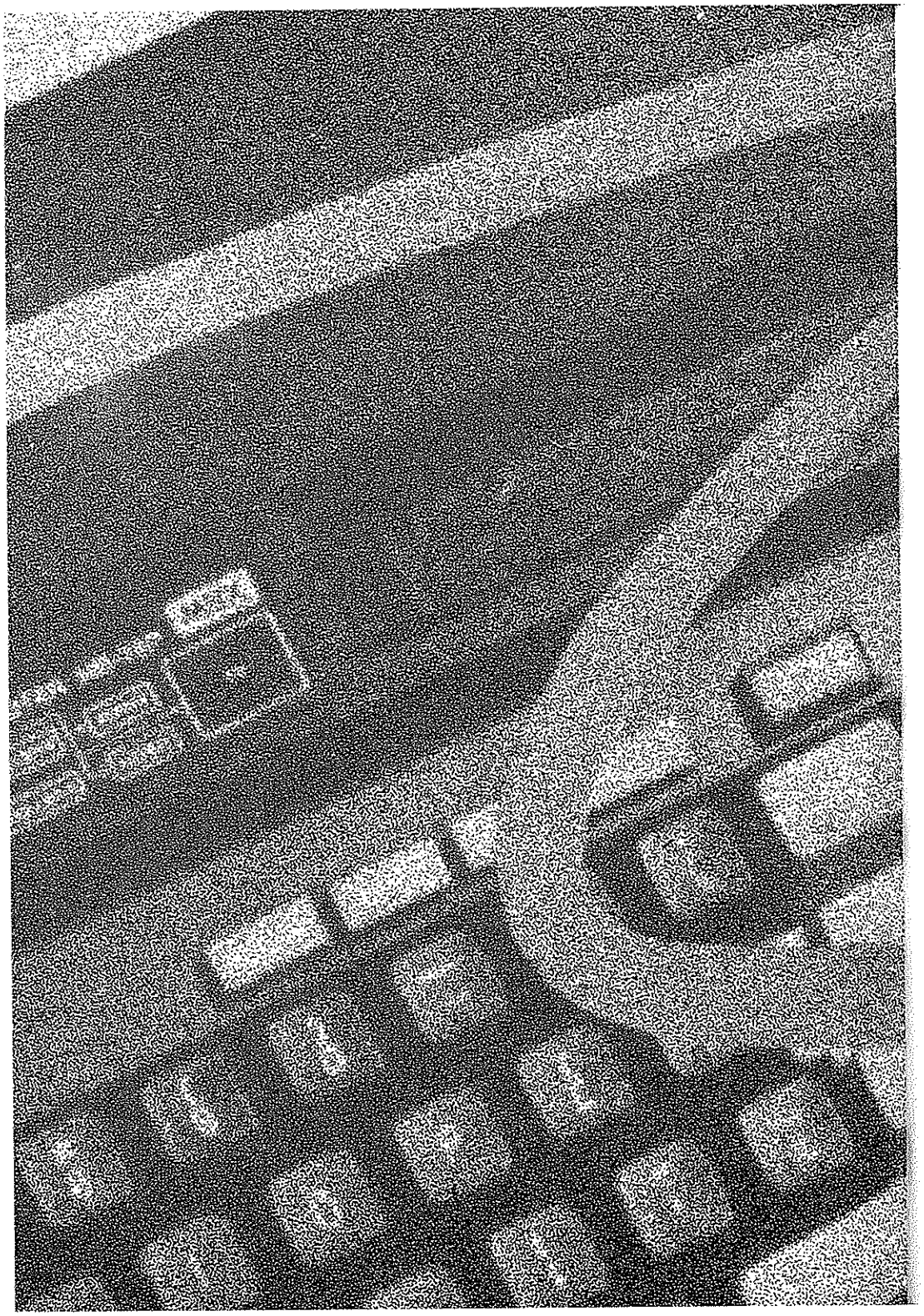
PARAMETROS DE SALIDA

Se actualizan algunos *flags* de detección (por ejemplo, ON SPRITE) y se cambia la tecla pulsada del *buffer* del teclado.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No se tocará el contenido de ningún registro, pero más de una posición de memoria se verá afectada.

Nota: Se llamará al vector H.KEYI después de apilar todos los registros, para poder procesar otra interrupción.



54

Puntos de entrada relacionados con el manejo de cartuchos

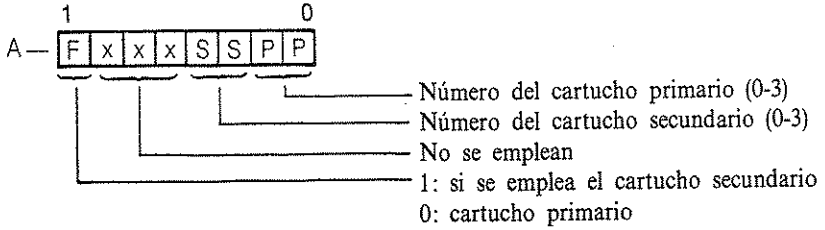
Las BIOS descritas en este tema soportan el sistema de cartuchos. (Véase la parte segunda, tema 50, "Memoria y cartucho del MSX".)

RDSLT 000C

Esta llamada se emplea para leer una posición de un cartucho.

PARAMETROS DE ENTRADA

El par de registros HL han de contener la dirección de la posición a leer, y el acumulador indicará de qué cartucho se lee. El contenido del acumulador tiene el siguiente significado: los dos bits menos significativos indicarán el cartucho primario a emplear (0-3); los dos bits siguientes indicarán el cartucho secundario (0-3); los tres bits que siguen no tienen significado alguno; el bit más significativo ha de estar a 1 para emplear el cartucho secundario (en caso contrario se empleará el primario).



PARAMETROS DE SALIDA

En el acumulador estará el contenido de la posición especificada.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera el contenido de los registros AF, BC y DE.

Nota: Esta rutina desactiva todo tipo de interrupciones. Si intentas acceder a la página 3 de un cartucho harás que el ordenador se bloquee. (Véase ENASLT para evitarlo.)

WRSLT 0014

Ejecución a través de CALL&H0014.

Se emplea para escribir en una posición determinada de un cartucho.

PARAMETROS DE ENTRADA

El par de registros HL contendrán la dirección de la posición a escribir del cartucho; el acumulador especificará el cartucho a emplear y el registro E tendrá el valor del byte a escribir. El contenido del acumulador tendrá el mismo formato que en la rutina RDSLT (000C).

PARAMETROS DE SALIDA

Esta rutina no devuelve parámetros.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada se alterarán los contenidos de los registros AF, BC y D.

Nota: La rutina descrita desactiva todo tipo de interrupciones. Si intentas acceder a la página 3 del cartucho se te bloqueará el ordenador. (Véase ENASLT para evitarlo.)

CALSLT 001C

Ejecución a través de CALL&H001C.

Esta rutina hace una llamada a un cartucho, que selecciona una página de otro cartucho y hace una llamada a una determinada dirección.

PARAMETROS DE ENTRADA

El registro IX contendrá la dirección a llamar, y el byte más significativo del registro de índice IY especificará el cartucho. El formato de este byte de IY es el mismo que el del acumulador en la RDSLT. Los parámetros a pasar pueden estar en los registros AF, BC, DE y HL, pero nunca en los registros alternativos.

PARAMETROS DE SALIDA

No se devuelve ningún parámetro, excepto si son devueltos por la llamada del cartucho; estos parámetros podrán devolverse en cualquier registro excepto en el acumulador alternativo, ya que éste contendrá el byte de selección de cartucho antes de la segunda llamada.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

El contenido de los registros AF', BC', DE' y HL' se cambiará antes de ejecutar la llamada al cartucho interno.

Nota: Esta rutina desactiva las interrupciones. Se podrá ejecutar a través de la instrucción RST6. (Véase CALLF, tema 29.) Si intentas acceder a la página 3 de un cartucho se te bloqueará el ordenador. (Véase ENASLT para evitarlo.)

ENASLT 0024

Ejecución a través de CALL&H0024

Esta llamada selecciona una página de un cartucho.

PARAMETROS DE ENTRADA

Los dos bits más significativos del registro H se emplean para seleccionar la página adecuada, del siguiente modo:

BMS	Página seleccionada
00	0000-3FFF
01	4000-FFFF
10	8000-BFFF
11	C000-FFFF

El acumulador indica el cartucho a seleccionar, según se explicó en RDSLT.

PARAMETROS DE SALIDA

No devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada altera los contenidos de AF, BC, DE y HL.

Nota: Esta rutina desactiva todo tipo de interrupciones.

Con esta llamada se evitarán los problemas de acceso a la página 3 en las instrucciones RDSLT, WRSLT y CALSLT. Por ejemplo, si quieres leer la dirección &HD000 en la página 3, debes emplear los siguientes códigos:

```
CALL   &H0138      ; LEE EL CONTENIDO DEL REGISTRO
                     DE SELECCION DEL CARTUCHO PRI-
                     MARIO.
PUSH   AF          ; APILA DICHO REGISTRO.
LD     HL;&HD000   ; DIRECCION A LEER.
PUSH   HL          ; APILA DICHA DIRECCION.
LD     A,3
DI                 ; DESACTIVA LAS INTERRUPCIONES, YA
                     QUE ESTAS AFECTAN A LA PAGINA 3.
CALL   &H0024      ; SELECCIONA LA PAGINA 3.
POP    HL          ; DESAPILA LA DIRECCION.
```

```
LD      H,(HL)          ; TOMA EL CONTENIDO DESEADO.
POP     AF
CALL    &H013B          ; VUELVE A ACTIVAR LA PAGINA 3 DEL
                          SISTEMA.
EI
LD      A,H             ; DEVUELVE EL VALOR DESEADO EN EL
                          ACUMULADOR.
RET
```

Se pueden emplear otros métodos parecidos para escribir y llamar subrutinas de la página 3 de otro cartucho.

RSLREG 0138

Ejecución a través &H0138.

Esta llamada lee el contenido del registro de selección del cartucho primario.

PARAMETROS DE ENTRADA

No necesita de estos parámetros.

PARAMETROS DE SALIDA

El acumulador contendrá una copia del registro de selección del cartucho primario.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo se cambia el valor contenido en el acumulador.

Nota: Esta rutina es una simple instrucción IN (IN&HA8 posiblemente), seguida de RET.

WSLREG 013B

Ejecución a través de CALL&H013B.

Con esta llamada escribimos en el registro de selección del cartucho primario.

PARAMETROS DE ENTRADA

El acumulador ha de tener el valor a escribir.

PARAMETROS DE SALIDA

Esta rutina no devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No afecta a ningún registro, ni tampoco a posición alguna.

Nota: Esta rutina es sencillamente una instrucción OUT (OUT&HA8 probablemente), seguida de RET.

CALBAS 0159

Ejecución a través de CALL&0159.

El intérprete de BASIC emplea este punto de entrada para llamar a una extensión de él mismo, contenida en un cartucho.

PARAMETROS DE ENTRADA

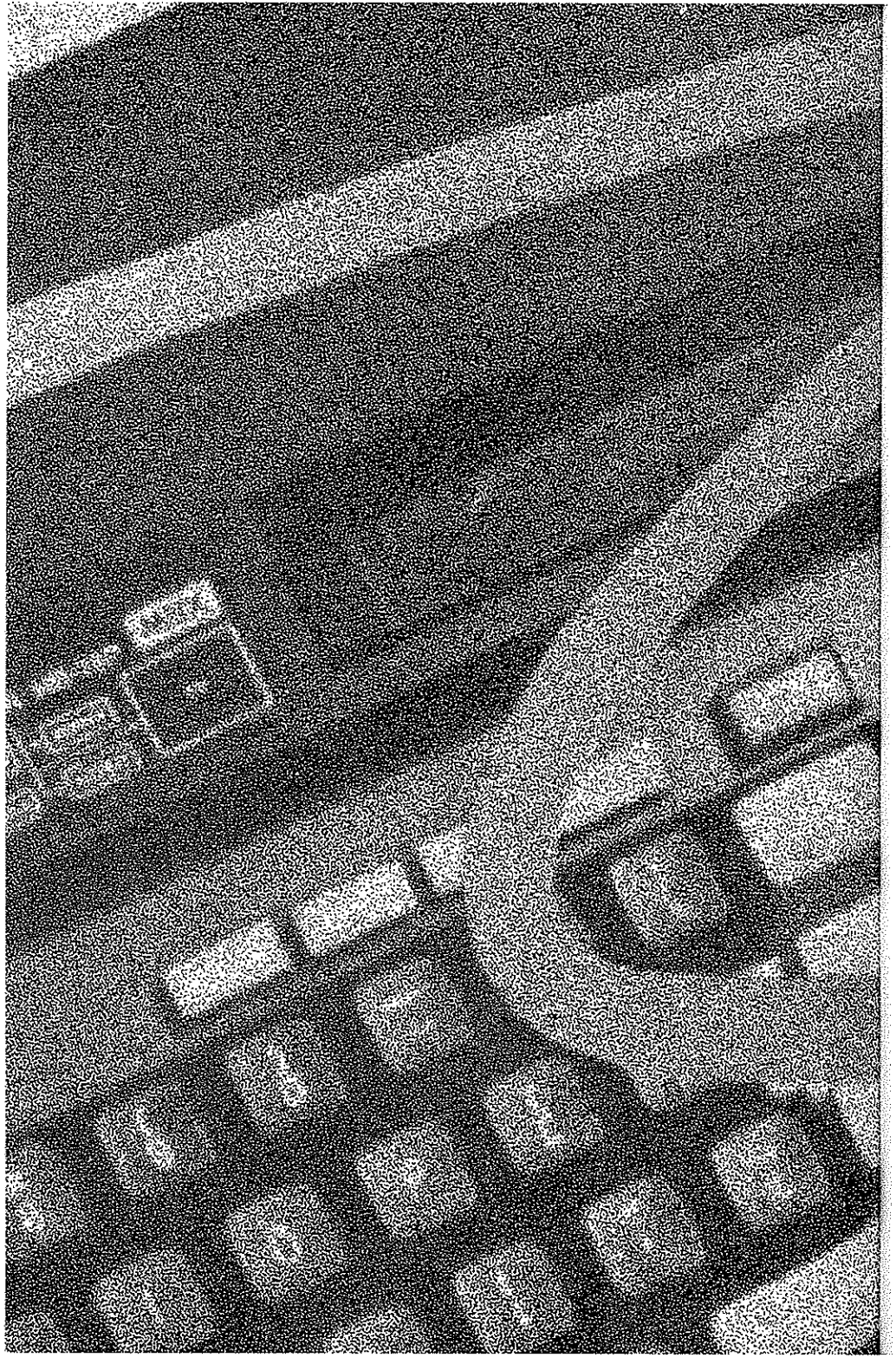
El registro IX ha de contener la dirección de llamada.

PARAMETROS DE SALIDA

La salida producida depende de la llamada al cartucho.

REGISTRO Y POSICIONES DE MEMORIA AFECTADOS

También dependen de la llamada.



55

Puntos de entrada empleados para acceder a la consola

Este tema describe las llamadas que controlan la "consola", o sea, el teclado, la pantalla en modo texto y la impresora.

CHSNS 009C

Ejecución a través de CALL &H009C.

Esta rutina lleva a cabo dos operaciones. Primero analiza el estado de las teclas *shift*. Si está pulsada y la visualización de las funciones de las teclas F1 ... F10 está activa, entonces muestra las funciones que realizan las teclas F6 ... F10 en pantalla. Esta rutina analiza también el estado del *buffer* del teclado.

PARAMETROS DE ENTRADA

No necesita de parámetros de entrada.

PARAMETROS DE SALIDA

Si el *buffer* del teclado está vacío, el *flag* Z se pone a 1; de otro modo estará a 0.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo afecta al acumulador y a los *flags*.

Nota: Esta rutina no inhibe las interrupciones.

CHGET 009F

Ejecución a través de CALL &H009F.

Esta rutina devuelve el carácter del *buffer* del teclado. Si el *buffer* del teclado está vacío espera a que se pulse una tecla, mostrando el cursor en pantalla si está activo.

PARAMETROS DE ENTRADA

No necesita de parámetros de entrada.

PARAMETROS DE SALIDA

El código de carácter de la tecla pulsada pasa al acumulador.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada sólo afecta a los *flags* y al acumulador.

Nota: Esta rutina hace una llamada al vector H.CHGE justamente después de apilar los pares de registros HL, DE y BC (en este orden). Con lo que se permite emplear otros periféricos.

CHPUT 00A2

Ejecución a través de CALL &H00A2.

Esta llamada saca un carácter a la consola, saltando a la siguiente línea o desplazando la pantalla si fuera necesario. Admite los caracteres de control del 7 al 13 y del 27 al 31, inclusive.

PARAMETROS DE ENTRADA

El código del carácter a enviar a consola debe estar en el acumulador. La posición del cursor en pantalla está en las posiciones CSRX y CSRY de memoria.

PARAMETROS DE SALIDA

A la salida la posición del cursor (CSRX, CSRY) habrá sido actualizada.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No cambia el contenido de ningún registro, pero altera los contenidos de las posiciones de memoria CSRX, CSRY y TTYPOS.

Nota: Después de apilar los registros HL, DE, BC y AF llama al vector H.CHPU, permitiendo escribir en otros periféricos (por ejemplo, una pantalla de ochenta columnas). Esta rutina no hará nada si están activos el modo gráfico 2 o el multicolor. No inhibe las interrupciones.

LPTOUT 00A5

Ejecución a través de CALL &H00A5.

Esta rutina envía un carácter a la impresora, si ésta estuviese conectada. Si la impresora no está preparada para recibir este carácter la rutina esperará a que esté lista o se pulse la tecla de parada STOP.

PARAMETROS DE ENTRADA

El carácter a pasar debe estar en el acumulador.

PARAMETROS DE SALIDA

Una vez terminado el *flag* de acarreo estará a 0 si el carácter fue enviado con éxito, y a 1 si se ha pulsado la tecla de parada para cortar esta operación.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada sólo afecta a los *flags*.

Nota: Lo primero que hace esta rutina es una llamada al vector H.LPTO, permitiendo que se ejecuten otros procesos. Un ejemplo sobre esto es programar un vector que ignore los caracteres que se envíen a la impresora, a través de

```
H.LPTO  INC  SP
        INC  SP
        RET
```

El vector H.LPTO comienza en la dirección &HFFB6, de modo que el ejemplo anterior puede ser programado desde el BASIC.

```
POKE &HFFB6,&H33
POKE &HFFB7,&H33
```

(Para hacer que funcione de nuevo la impresora ejecuta POKE &HFFB6,&HC9.)

LPTSTT 00A8

Ejecución a través de CALL &00A8.

Esta rutina analiza si la impresora está lista para recibir un nuevo carácter.

PARAMETROS DE ENTRADA

No necesita de ellos.

PARAMETROS DE SALIDA

Si la impresora está lista el acumulador tendrá 255 y el *flag* Z estará a 0; de otro modo, el acumulador contendrá 0 y el *flag* Z estará a 1.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo serán afectados los *flags* y el acumulador.

Nota: La rutina LPTOUT llama a esta rutina, que lo primero que hace es llamar, asimismo, al vector H.LPTS.

CNVCHR 00AB

Ejecución a través de CALL &H00AB.

Con esta llamada el código de un carácter se transforma en el del número de modelo que representa en el procesador de video. Para los códigos de carácter entre 32 y 255 este número es igual al número de modelo de dicho carácter, ya

que los códigos de 0 a 31 corresponden a caracteres de control; sin embargo, los números de diseño de 0 a 31 representan los caracteres que se obtienen pulsando la tecla gráfica (GRAPH). Estos diseños están representados por dos códigos de carácter: el código de control 1, seguido de un carácter con un código entre 64 y 95.

PARAMETROS DE ENTRADA

En el acumulador debe estar el carácter a transformar.

PARAMETROS DE SALIDA

Se pueden obtener cuatro tipos de resultados según el estado del byte gráfico (GRPHED) y el acumulador:

1. Si GRPHED tiene un cero y el acumulador tiene el código del carácter de control 1, el acumulador no variará, GRPHED se pondrá a 1 y los *flags* de acarreo y cero se pondrán a 0.
2. Si GRPHED tiene un cero y en el acumulador hay cualquier valor distinto de 1, entonces el contenido del acumulador no variará, pero los *flags* C y Z se pondrán a 1.
3. Si GRPHED tiene un valor distinto de cero y el contenido del acumulador está entre 64 y 95, el contenido del acumulador será 64 unidades menor que el valor inicial, GRPHED tendrá un cero, el *flag* C se pondrá a 1 y el *flag* Z a 0.
4. Si el contenido de GRPHED es distinto de cero y el contenido del acumulador no está comprendido entre 64 y 95, GRPHED se pondrá a 0 y los C y Z se pondrán 1, pero el contenido del acumulador no variará.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada variarán el acumulador, los *flags* y la posición GRPHED.

PINLIN 00AE

Ejecución a través de CALL &H00AE.

Esta rutina es similar a INLIN; pero, estando en el modo de autonumeración de líneas, si pulsas <CTRL-U> no borrará el número de línea.

PARAMETROS DE ENTRADA

No necesita de parámetros de entrada.

PARAMETROS DE SALIDA

El registro HL apunta a la siguiente dirección después del primer carácter del *buffer;* el *flag* de acarreo estará a 1 si se pulsó <CTRL><STOP>.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada se altera el contenido de los registros AF, BC, DE y HL.

INLIN 00B1

Ejecución a través de CALL &H00B1.

El intérprete de BASIC emplea esta rutina para tomar una línea del teclado, mostrando los caracteres en pantalla según se teclean y almacenándolos en el *buffer* hasta que pulse <RETURN> o <CTRL><STOP>.

PARAMETROS DE ENTRADA

No necesita ningún parámetro de entrada.

PARAMETROS DE SALIDA

El registro HL apuntará al carácter siguiente al primero del *buffer;* el *flag* de acarreo estará a 1 si se pulsó <CTRL><STOP>.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada varía los contenidos de AF, BC, DE y HL.

QUINLIN 00B4

Ejecución a través de CALL &H00B4.

Es una rutina parecida a INLIN: visualiza la interrogación de las instrucciones INPUT y se prepara para aceptar una línea del teclado.

PARAMETROS DE ENTRADA

No necesita parámetros de entrada.

PARAMETROS DE SALIDA

El registro HL apunta al carácter siguiente al primero en el *buffer;* el *flag* de acarreo estará a 1 si pulsaste <CTRL><STOP>.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada afectará al contenido de los registros AF, BC, DE y HL.

BREAKX 00B7

Ejecución a través de CALL &H00B7.

Esta rutina analiza si se ha pulsado <CTRL><STOP>.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

Si se pulsó <CTRL><STOP> el *flag* de acarreo estará a 1; de otro modo será 0.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada varía los contenidos del acumulador y los *flags.*

Nota: Necesita que las interrupciones estén desactivadas.

ISCNTC 00BA

Ejecución a través de CALL &H00BA.

Analiza si se han pulsado <STOP> o <CTRL><STOP>. Si se pulsó <STOP> el programa se detiene, manteniéndose hasta que se pulse otra vez

<STOP> o <CTRL><STOP>. Si se pulsa <CTRL><STOP> y no ha interrumpido, el ordenador pasa la pantalla al modo texto y al modo directo del BASIC. Si no se pulsa ninguna de estas opciones la rutina no hará nada.

PARAMETROS DE ENTRADA

No los necesita; sin embargo, INTFLG tendrá tres si se pulsó <CTRL><STOP>, o cuatro si no se presionó <STOP>, debido a las interrupciones de 50 Hz.

PARAMETROS DE SALIDA

No devuelve valor alguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Afecta al acumulador, los *flags* y las posiciones de memoria INTFLG y KILBUF.

Nota: lo primero que hace esta rutina es analizar la posición BASROM. Si su contenido es cero la rutina efectúa lo dicho anteriormente; si no, devuelve el control. Por tanto, si pones un valor distinto de cero en BASROM evitarás que sean interrumpidos tus programas (empléalo con cuidado).

Las interrupciones han de estar activadas para emplear esta rutina.

CKCNTC 00BD

Ejecución a través de CALL &H00BD.

Esta rutina hace exactamente lo mismo que ISCNTC, pero algo más despacio; por ello, te recomendamos que emplees ISCNTC.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

Esta llamada no devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Varían los contenidos del acumulador, *flags* y posiciones de memoria INTFLG y KILBUF.

BEEP 0C00

Ejecución a través de CALL &H0C00.

Con esta llamada se emite un sonido (BEEP) cada vez que pulses <CTRL><G>.

PARAMETROS DE ENTRADA

Con esta llamada no se necesitan estos parámetros.

PARAMETROS DE SALIDA

Tampoco devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina afectará a los registros AF, BC, DE, HL y a las posiciones de memoria MUSICF, PLYCNT; de VCBA, a la VCBA+4; de VCBB, a la VCBB+4, y de VCBC, a la VCBC+4.

Nota: Al terminar esta rutina hace un salto a la BIOS GICINI, que inicializa el generador de sonido.

CLS 00C3

Ejecución a través de CALL &H00C3.

Con esta llamada borramos la pantalla, incluso estando en modo gráfico.

PARAMETROS DE ENTRADA

El *flag* Z ha de estar a 0 para borrar la pantalla; si estuviera a 1 no la borraría.

PARAMETROS DE SALIDA

no devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Varían los contenidos de los registros AF, BC, DE y las posiciones de memoria LINTTB, CSRX y CSRY.

POSIT 00C6

Ejecución a través de CALL &H006C.

Esta rutina lleva el cursor hasta una posición determinada.

PARAMETROS DE ENTRADA

El número de columna debe estar en el registro H, y el número de fila, en el registro L.

PARAMETROS DE SALIDA

Las posiciones de memoria CSRX y CSRY se actualizarán.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada varían los contenidos del registro AF y las posiciones CSRX y CSRY.

FNKSB 00C9

Ejecución a través de CALL &H00C9

Esta llamada analiza si está activa la visualización de las funciones de las teclas F1 ... F10, y si es así muestra sus funciones en pantalla.

PARAMETROS DE ENTRADA

Si CNSDFG contiene un cero, esta llamada no hará nada; si es distinto de cero, llama a DSPFNK.

PARAMETROS DE SALIDA

no devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Los contenidos de los registros AF, BC y DE se verán alternados.

Nota: Esta rutina no inhibe las interrupciones.

ERAFNK 00CC

Ejecución a través de CALL &H00CC.

Con esta llamada se borran las definiciones de las teclas de función en la pantalla, si está en uno de los modos de texto.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

No devuelve ninguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Los únicos registros que quizá sean alterados serán AB, BC y CD.

DSPFNK 00CF

Ejecución a través de CALL &H00CF.

Con esta llamada se escriben las funciones de las teclas F1 ... F10 en la parte inferior de la pantalla.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

A la posición CNSDFG se le da el valor 255, indicando así que la definición de las teclas de función se muestra por pantalla.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada variará los contenidos de los registros AF, BC y DE, así como de la posición de memoria CNSDFG.

Nota: Esta rutina no inhibe las interrupciones.

TOTEXT 00D2

Ejecución a través de CALL &H00D2.

Esta rutina pone la pantalla en uno de los modos de texto, estando ésta en otro modo diferente.

PARAMETROS DE ENTRADA

La posición de memoria OLDSCR contendrá el modo de texto en que se ha puesto la pantalla.

PARAMETROS DE SALIDA

No devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada cambian los contenidos de los registros AF, BC, DE y HL, así como las proporciones de memoria LINLEN, NAMBAS, CGPBAS y SCRMOD.

Nota: El VDP no puede estar en ningún modo de texto. En esta rutina se hace una llamada al vector H-TOTE, pasándole el contenido de OLDSCR en el acumulador. Después de ejecutar dicho vector se cede el control al punto de entrada del BIOS CHGMOD.

Esta rutina no inhibe las interrupciones.

CHGCAP 0132

Ejecución a través de CALL &0132.

Esta rutina controla el estado de la luz indicadora de la tecla <CAPS LOCK>.

PARAMETROS DE ENTRADA

Si el acumulador contiene un cero se apagará la luz; en cambio, si su contenido es distinto de cero se encenderá.

PARAMETROS DE SALIDA

Esta rutina no devuelve parámetros.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada sólo se alteran los contenidos del acumulador y de los *flags*.

SNSMAT 0141

Ejecución a través de CALL &H0141.

Con esta llamada se analiza el estado de una de las columnas de la matriz del teclado y devuelve el estado de dichas teclas.

PARAMETROS DE ENTRADA

El número de columna a analizar se pasará a través del acumulador.

PARAMETROS DE SALIDA

El estado de las teclas de dicha columna se volcará en el acumulador; si se está pulsando una tecla, entonces el bit correspondiente estará 0; las teclas que no se pulsen estarán a 1.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada sólo modificará los *flags* y el acumulador.

Nota: No inhibe las interrupciones.

ISFLIO 014A

Ejecución a través de CALL &H014A.

Esta rutina analiza si hay alguna unidad de E/S en transferencia de datos.

PARAMETROS DE ENTRADA

No los emplea.

PARAMETROS DE SALIDA

Si no hay transferencia con ningún dispositivo de E/S, entonces el acumulador contendrá cero y el *flag* Z estará a 1; de otro modo el *flag* Z estaría a 0 y en el acumulador habrá un valor distinto de 0.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada sólo modifica los contenidos del acumulador y los *flags*.

OUTDLP 014D

Ejecución a través de CALL &H014D.

Con esta llamada el intérprete de BASIC escribe caracteres en la impresora.

PARAMETROS DE ENTRADA

El carácter a escribir deberá estar almacenado en el acumulador.

PARAMETROS DE SALIDA

Esta rutina no devuelve ningún parámetro, pero si esta operación se rompe se cederá el control a la parte del intérprete de BASIC dedicada al tratamiento de errores, y se obtendrá un mensaje de error de unidad de E/S *(Device I/O Error)*.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo afectará a los *flags*.

Nota: Esta rutina transforma la función TAB en los espacios correspondientes.

KILBUF 0156

Ejecución a través de CALL &H0156.

Con esta llamada se borra el *buffer* del teclado.

PARAMETROS DE ENTRADA

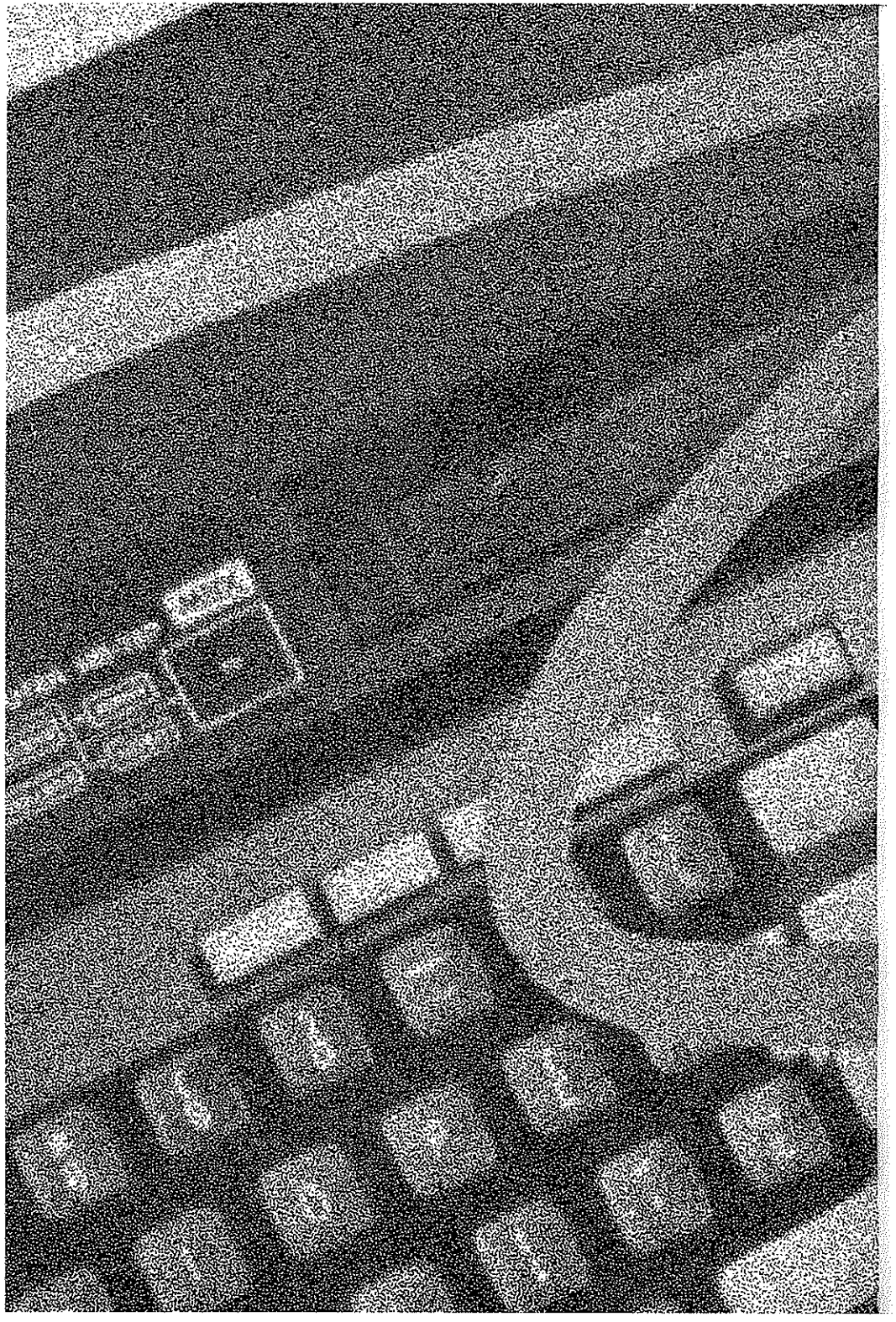
No se necesitan.

PARAMETROS DE SALIDA

Tampoco devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada sólo varía el contenido del registro HL.



56

BIOS de control de los puertos de los *joysticks*

Las siguientes rutinas llevan el control de los puertos del *joystick* y otros periféricos que se conecten a ellos.

GTSTCK 00D5

Ejecución a través de CALL & H00D5.

Esta llamada devuelve el estado del cursor o de uno de los *joysticks*.

PARAMETROS DE ENTRADA

En el acumulador debe estar el identificador de *joysticks*, que será 0 para las teclas de cursor, 1 para el *joystick* 1 y 2 para el *joystick* 2.

PARAMETROS DE SALIDA

La dirección de movimiento del *joystick* (o de las teclas de cursor) se devuelve a través del acumulador. Las direcciones son:

0 ningún movimiento.
1 hacia arriba.
2 hacia arriba a la derecha.
3 hacia la derecha.
4 hacia abajo a la derecha.
5 hacia abajo.
6 hacia abajo a la izquierda.
7 hacia la izquierda.
8 hacia arriba a la izquierda.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Los únicos contenidos que varían son los de los registros AF, BC, DE y HL.

Nota: Esta llamada no inhibe las interrupciones.

GTTRIG 00D8

Ejecución a través de CALL &H00D8.

Esta llamada devuelve el estado de la barra espaciadora o de algún disparador de los *joysticks*.

PARAMETROS DE ENTRADA

En el acumulador se indica qué disparador se tiene que analizar. Los valores serán:

0 = → barra espaciadora.
1 = → disparador 1A.
2 = → disparador 2A.
3 = → disparador 1B.
4 = → disparador 2B.

PARAMETROS DE SALIDA

El contenido del acumulador será 255, si se está pulsando el disparador; en caso contrario contendrá cero.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo varían los contenidos de los registros AF, BC, DE y HL.

Nota: Esta rutina no inhibe las interrupciones.

GTPAD 00DB

Ejecución a través de CALL &00DB.

Esta llamada devuelve el estado de uno de los tableros gráficos conectado a uno de los puertos del *joystick*.

PARAMETROS DE ENTRADA

En el acumulador habrá un valor de 0 a 7, dependiendo de la información que necesitemos. Si el contenido del acumulador está entre 0 y 3, el estado devuelto es del puerto 1 del *joystick*; si está entre 4 y 7, será el puerto 2. Para saber si se está pulsando el tablero gráfico toma los valores 0 y 4; para la coordenada X emplea los valores 1 y 5, y mete 2 y 6 para la coordenada Y. Para saber si se está pulsando el *switch* mete 3 ó 7 en el acumulador.

PARAMETROS DE SALIDA

El estado del tablero gráfico se devuelve en el acumulador. Para el caso de haber un 0 o un 4 en dicho acumulador se devolverá un 0 si el tablero correspondiente se está pulsando; de otro modo habrá 255. Lo mismo ocurre con el *switch* (valores 3 y 7 en el acumulador). Si los parámetros pasados en el acumulador

eran 1 ó 2 (5 ó 6 para el *joystick* 2) éste devolverá las coordenadas X o Y del punto, respectivamente.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada sólo llegará a alterar los contenidos de los registros AF, BC, DE y HL.

Nota: Esta llamada no desactiva las interrupciones. Para más información consulta la instrucción PAD del BASIC.

GTPDL 00DE

Ejecución a través de CALL &H00DE.

Con esta llamada sabremos el estado de uno de los posibles *paddles* conectados a los puertos del *joystick*.

PARAMETROS DE ENTRADA

El número del *paddle* lo has de pasar a través del acumulador. Será un número impar para los *paddles* conectados al puerto 1 del *joystick*, y un número par para los conectados al puerto 2.

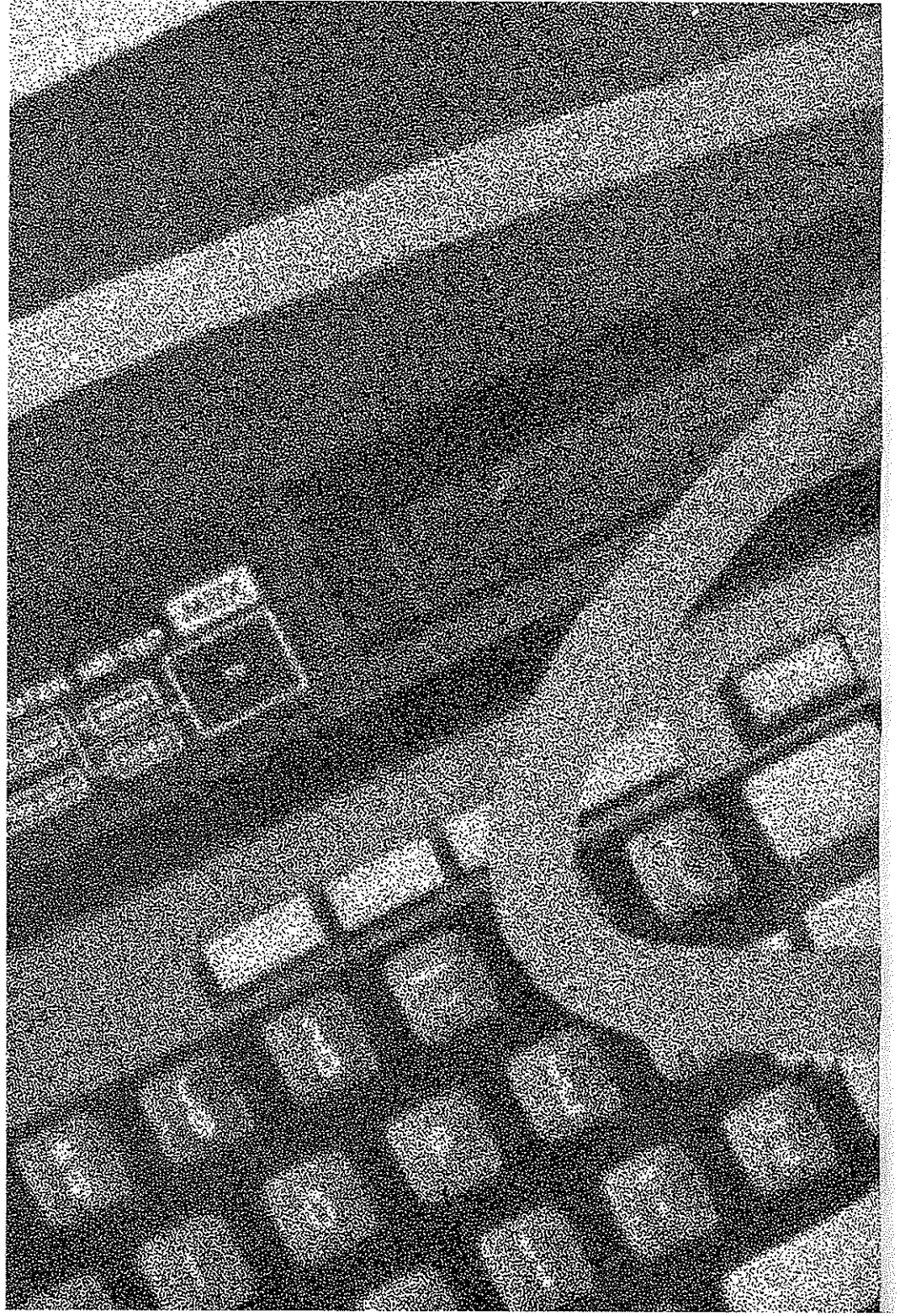
PARAMETROS DE SALIDA

El acumulador devolverá un número entre 0 y 255, indicando el estado del *paddle* pedido.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Los únicos registros que pueden ser alterados son AF, BC, DE y HL.

Nota: Esta llamada no inhibe las interrupciones.



57

Llamadas a BIOS relacionadas con el interfaz del cassette

Las llamadas descritas en esta sección sirven para controlar el interfaz del cassette.

TAPION 00E1

Ejecución a través de CALL &H00E1.

Esta rutina pone en marcha el cassette y lee la cabecera de la cinta.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

Si se corta la operación el *flag* de acarreo se pone a 1.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Pueden variar los contenidos de los registros AF, BC, DE y HL.

Nota: Mientras se lee del cassette se inhibirán las interrupciones.

TAPIN 00E4

Ejecución a través de CALL &H00E4.

Con esta instrucción leemos un byte de la cinta.

PARAMETROS DE ENTRADA

No necesita parámetros de entrada.

PARAMETROS DE SALIDA

El acumulador contendrá el dato leído. Si se corta la operación se pondrá a 1 el *flag* de acarreo.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Los contenidos de los registros AF, BC, DE y HL pueden variar con esta llamada.

Nota: Las interrupciones se inhibirán mientras se lea del cassette.

TAPIOF 00E7

Ejecución a través de CALL &H00E7.

Esta rutina detiene la lectura del cassette.

PARAMETROS DE ENTRADA

No emplea estos parámetros.

PARAMETROS DE SALIDA

Tampoco los necesita.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No afecta a ningún registro ni a ninguna posición de memoria.

TAPOON 00EA

Ejecución a través de CALL &00EA.

Esta llamada pone en marcha el cassette y escribe la cabecera en la cinta.

PARAMETROS DE ENTRADA

Si el acumulador tiene un cero la cabecera será corta; si es distinto de cero escribirá una cabecera larga.

PARAMETROS DE SALIDA

Si se detiene la operación se pondrá a 1 el *flag* de acarreo.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

El contenido de los registros AF, BC, DE y HL puede variar con esta llamada.

Nota: Esta rutina no inhibe las interrupciones.

TAPOUT 00ED

Ejecución a través de CALL &H00ED.

Esta rutina escribe un byte en la cinta de cassette.

PARAMETROS DE ENTRADA

El acumulador debe contener el byte a escribir.

PARAMETROS DE SALIDA

Si se corta la operación el *flag* de acarreo repondrá a 1 a la salida.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina puede modificar los contenidos de los registros AF, BC, DE y HL.

Nota: Esta llamada no inhibe las interrupciones.

TAPOOF 00F0

Ejecución a través de CALL &H00F0.

Esta rutina detiene la escritura en cinta.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

Tampoco devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No altera el contenido de ningún registro ni posición de memoria.

STMOTR 00F3

Ejecución a través de CALL &H00F3.

Con esta rutina se conecta o desconecta el motor del cassette, con la opción de hacerlo de forma alternativa.

PARAMETROS DE ENTRADA

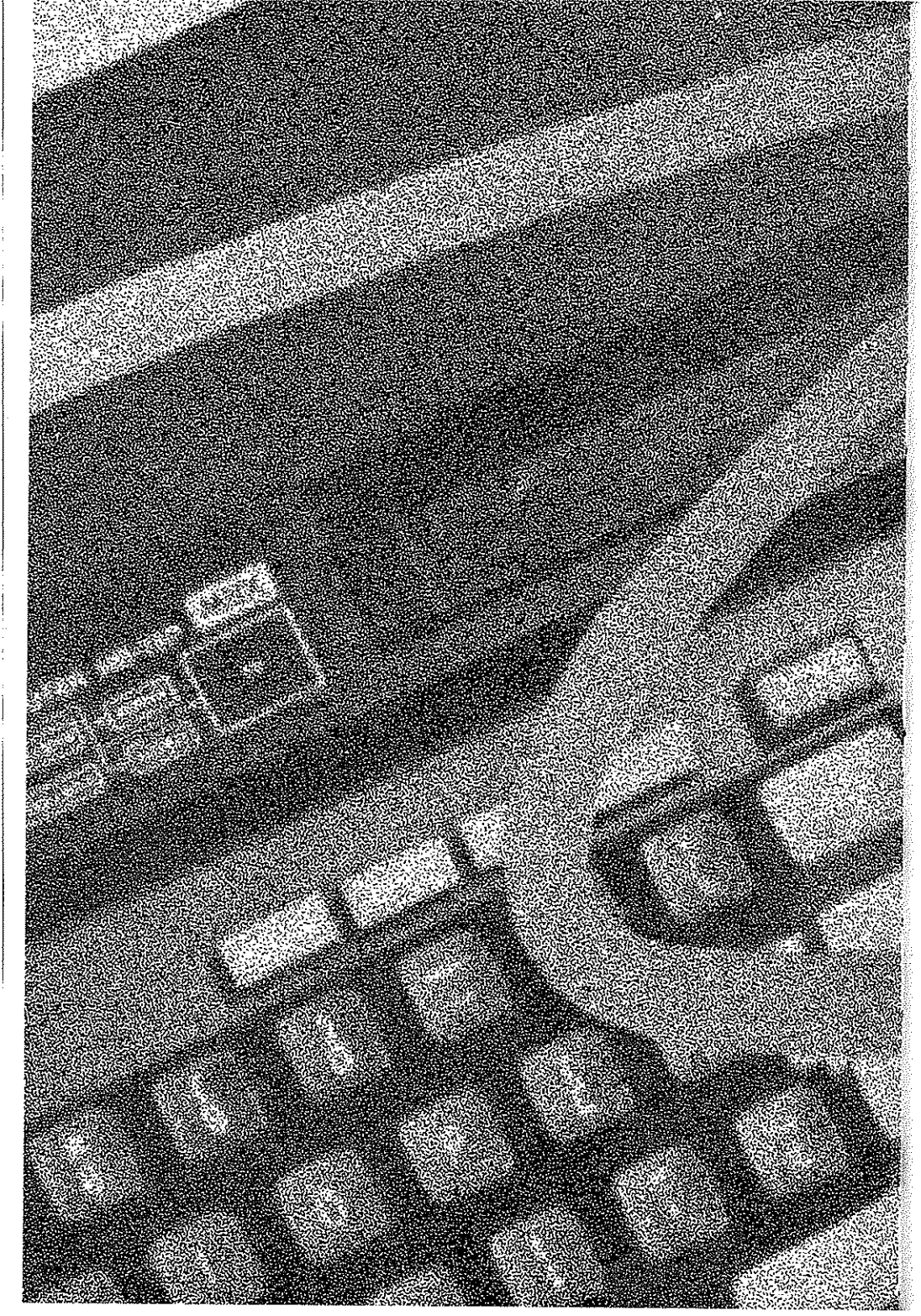
Si el contenido del acumulador es 1 el motor del cassette se conecta; si vale 0 se desconecta; si es 255, entonces se irá intercambiando conexión con desconexión cada vez que ejecutemos esta rutina.

PARAMETROS DE SALIDA

No devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo altera el contenido del acumulador y los *flags*.



58

Puntos de entrada relacionados con el sonido

Las BIOS siguientes controlan el Generador de Sonido Programable (PSG).

GICINI 0090

Ejecución a través de CALL &H0090.

Con esta llamada se inicializa el generador de sonido programable, nivelando las escalas, apagando así cualquier sonido en emisión.

PARAMETROS DE ENTRADA

No necesita ninguno de estos parámetros, pero deberás desactivar las interrupciones antes de llamar a esta rutina.

PARAMETROS DE SALIDA

No devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No se altera el contenido de ningún registro, pero las posiciones de memoria MUSICF, PLYCNT; de VCBA, a VCBA+4; de VCCBB, a VCBB+4, y de VCBC, a VCBC+4, se pondrán a cero.

WRTPSG 0093

Ejecución a través de CALL &H0093.

Esta rutina escribe un valor en uno de los registros del generador de sonido programable.

PARAMETROS DE ENTRADA

En el acumulador estará el número del registro a escribir, con un valor entre 0 y 13. El dato a escribir estará en el registro E.

PARAMETROS DE SALIDA

No devuelve valor alguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No altera el contenido de ningún registro ni posición de memoria.

Nota: Las interrupciones se desactivan al comienzo de esta rutina y se volverán a activar a la salida.

RDPSG 0096

Ejecución a través de CALL &H0096.

Con esta llamada se lee el contenido de uno de los registros del PSG.

PARAMETROS DE ENTRADA

En el acumulador estará el número del registro a leer, que debe estar comprendido entre 0 y 13.

PARAMETROS DE SALIDA

El contenido del registro se devolverá en el acumulador.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo se verá alterado el contenido del acumulador.

Nota: Las interrupciones se desactivan al comienzo de esta rutina y se vuelven a activar a la salida.

STRTMS 0099

Ejecución a través de CALL &H0099.

Esta rutina ejecuta la música si hay una escala escrita para ella.

PARAMETROS DE ENTRADA

No los tiene.

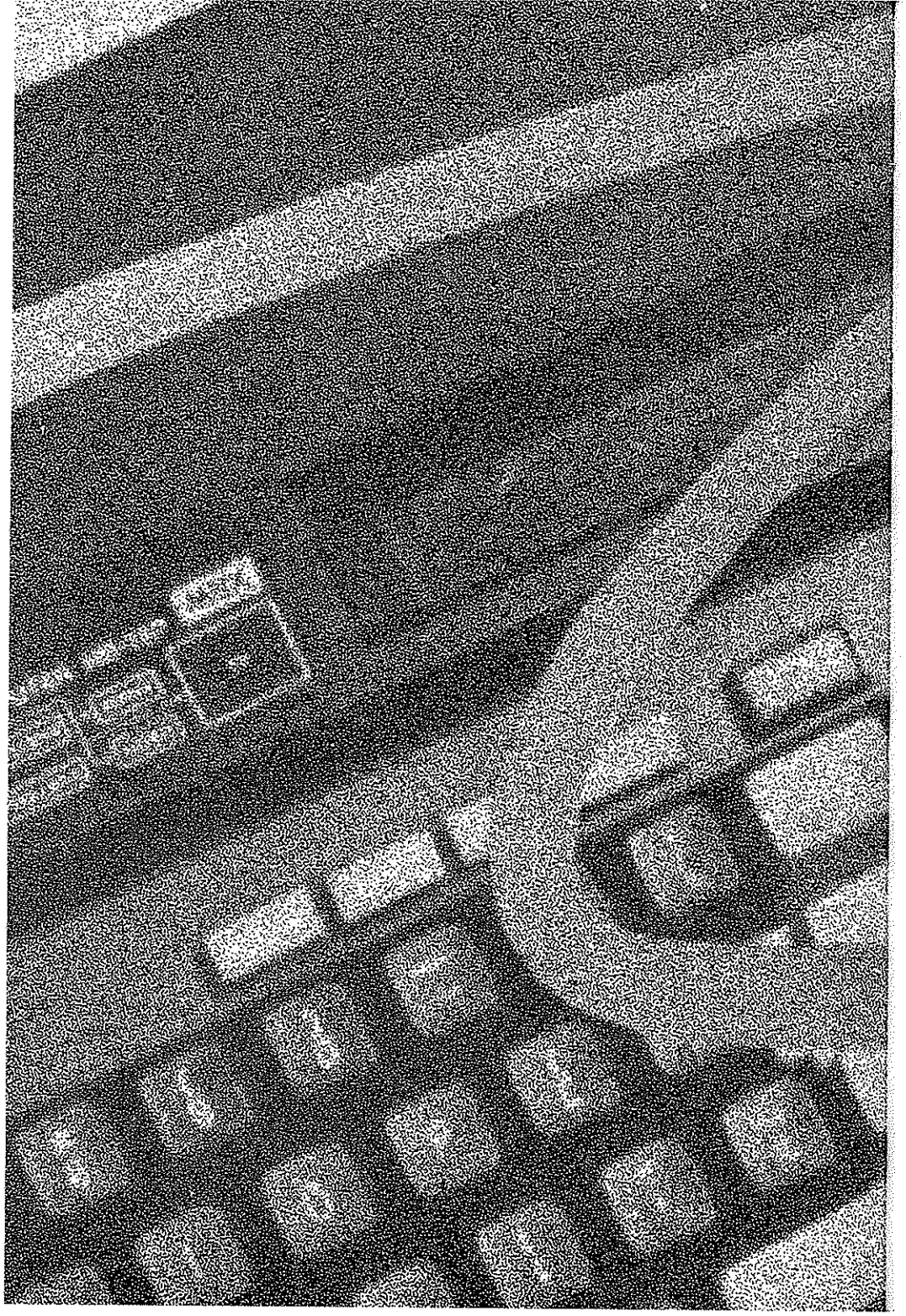
PARAMETROS DE SALIDA

Si el *buffer* de sonido está vacío el acumulador contendrá cero.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada altera los contenidos de las posiciones de memoria PLYCNT y MUSICF, además de los registros AF y HL.

Nota: La música no comenzará a ejecutarse hasta la siguiente interrupción de 50 Hz.



59

Puntos de entrada relacionados con el VDP

Las BIOS de este tema llevan el control del procesador de video del MSX.

DISSCR 0041

Ejecución a través de CALL &H0041.

Desactiva la pantalla y ocasiona que se ponga en el color de fondo. Todo lo que teclees se enviará a la pantalla, pero no lo verás hasta que se llame a ENASCR (&H0044). Cambiando el modo de pantalla también activarás la pantalla.

PARAMETROS DE ENTRADA

Ninguno.

PARAMETROS DE SALIDA

Ninguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina modifica el contenido de los registros AF y BC; no inhibe las interrupciones.

POSIBLES USOS

Puedes emplear esta rutina desde el BASIC a través de la función USR, borrando así la pantalla. Empléala para hacer un dibujo y, al volver a activar la pantalla (ENASCR), dará la impresión de que ha sido instantáneo. Ejemplo:

```
10 DEF USR0 = &H41.
20 DEF USR1 = &44.
30 CLS.
40 REM DESACTIVA LA PANTALLA.
50 X = USRO (0).
60 INPUY PW$.
70 REM ACTIVA LA PANTALLA.
80 X = USR1 (0).
```

ENASCR 0044

Ejecución a través de CALL &H0044.

Se llama a esta subrutina para volver a activar la pantalla, después de haberla desactivado con DISSCR.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

No devuelve ninguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera los contenidos de los registros AF y BC; no inhibe las interrupciones.

POSIBLES USOS

Activa la pantalla después de DISSCR. En DISSCR tienes un ejemplo.

WRTVDP 0047

Ejecución a través de CALL &H0047.

Con esta llamada escribimos en uno de los registros del VDP. Tienes más detalles de los efectos de esta rutina en la *Guía avanzada de programación*, tema 16, en la parte del procesador de video.

PARAMETROS DE ENTRADA

El número del registro a escribir ha de estar en el registro C, y el valor debe ser escrito en el registro B.

PARAMETROS DE SALIDA

No devuelve ningún parámetro, pero el valor especificado en B estará en la posición de memoria RG0SAV+C.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada modifica los contenidos de los registros AF y BC, además de la posición de memoria RGXSAV, siendo X el registro escrito. Por ejemplo, si C tiene un 5 y B tiene un 0, entonces a la salida de la rutina la posición RG5SAV tendrá un 0. Esta rutina no desactiva las interrupciones.

POSIBLES USOS

Es un punto de entrada muy potente que controla el VDP. Te recomendamos

que emplees esta rutina para escribir en los registros del VDP, ya que guarda una copia de lo escrito. Cuando metes un valor en un registro no hay modo de conocerlo, ya que son registros "exclusivos de escritura".

RDVDP 013E

Ejecución a través de CALL &H013E.

Esta rutina se emplea para leer el registro de estado del procesador de video.

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

A la salida tendremos en el acumulador una copia del registro del VDP.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo cambia el valor del acumulador.

Nota: Esta llamada no cambia el estado de las interrupciones.

RDVRM 004A

Ejecución a través de CALL &H004A.

Esta rutina lee el contenido de una posición de la RAM de video.

PARAMETROS DE ENTRADA

La dirección de la posición a leer debes ponerla en el par de registros HL.

PARAMETROS DE SALIDA

En el acumulador tendrás el contenido de la posición apuntada por HL. Esta rutina no inhibe las interrupciones. El estado de los *flags* no está influenciado por el contenido del acumulador.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina sólo cambia los contenidos del acumulador y los *flags*.

WRTVRM 004D

Ejecución a través de CALL &004D.

Con esta llamada puedes escribir el contenido del acumulador en una posición determinada de la RAM de video.

PARAMETROS DE ENTRADA

En el par de registros HL estará la dirección de la posición a escribir, y en el acumulador el valor a escribir.

PARAMETROS DE SALIDA

Ninguno.

POSIBLES USOS

Para el acceso a la RAM de video.

SETRD 0050

Ejecución a través de CALL &H0050.

Con esta llamada preparamos el procesador de video para una operación de lectura.

PARAMETROS DE ENTRADA

En el par del registro HL debe estar la dirección a leer.

PARAMETROS DE SALIDA

Ninguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera los contenidos del acumulador y los *flags*. No inhibe las interrupciones.

POSIBLES USOS

Antes de leer en la RAM de video hemos de preparar el procesador de video para dicha operación. Las rutinas RDVRM y LDIRMV hacen una llamada a esta rutina; por tanto, no te será necesario emplearla, a no ser que quieras leer directamente.

SET WRT 0053

Ejecución a través de CALL &H0053.

Esta rutina prepara el procesador de video para una operación de lectura.

PARAMETROS DE ENTRADA

En el par de registros HL debe estar la dirección a leer de la RAM de video.

PARAMETROS DE SALIDA

Ninguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera el contenido del acumulador y de los *flags*. No inhibe las interrupciones.

POSIBLES USOS

No necesitas llamarla a no ser que quieras leer directamente. Las rutinas WRTVRM, FILVRM y LDIRVM hacen una llamada a esta rutina.

FILVRM 0056

Ejecución a través de CALL &H0056.

Esta rutina escribe un mismo valor en una sección de la RAM de video.

PARAMETROS DE ENTRADA

En el registro HL debes poner la dirección del primer byte de la sección, en BC la longitud de la sección y en el acumulador el valor a escribir.

PARAMETROS DE SALIDA

Esta rutina no devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera el contenido de los registros AF y BC. No inhibe las interrupciones.

POSIBLES USOS

Es bastante útil para poner parte de la pantalla de un mismo color y de manera rápida. La instrucción PAINT del BASIC la emplea.

LDIRMV 0059

Ejecución a través de CALL &H0059.

Con esta llamada pasamos el contenido de un bloque de la RAM de video a la memoria principal.

PARAMETROS DE ENTRADA

En HL estará la dirección de comienzo del bloque de la RAM de video, en BC estará su longitud y en DE la dirección de memoria principal a partir de la cual meteremos el bloque.

PARAMETROS DE SALIDA

El resultado será un bloque en memoria a partir dc DE y con la longitud de BC.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

AF, BC, DE y el bloque de destino en memoria será únicamente lo alterado. No inhibe las interrupciones.

POSIBLES USOS

Esta rutina resulta muy útil en los procesos de manipulación de la pantalla, ya que toma un bloque entero, lo procesa y lo devuelve a la RAM de video con LDIRVM. Otra utilidad es la de guardar una pantalla en memoria principal y en el momento que la necesitemos pasarla a la RAM de video con LDIRVM.

Nota: No confundas esta rutina con LDIRVM (LDIRMV lee un bloque "de" la RAM de video).

LDIRVM 005C

Ejecución a través de CALL &H005C.

Esta rutina lleva un bloque de la memoria principal a la RAM de video.

PARAMETROS DE ENTRADA

La dirección de comienzo del bloque debe estar en HL, y la de destino de la RAM de video, en DE; BC tendrá la longitud del bloque.

PARAMETROS DE SALIDA

Esta rutina no devuelve parámetros.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera los contenidos de AF, BC y DE; no inhibe las interrupciones.

Nota: No la confundas con LDIRMV (LDIRVM escribe un bloque "en" la RAM de video).

CHGMOD 005F

Ejecución a través de CALL &H005F.

Esta rutina pone al VDP en uno de sus cuatro modos de operación, o sea, en modo texto, en modo gráfico I (32 columnas), en modo gráfico II o en modo multicolor.

PARAMETROS DE ENTRADA

En el acumulador ha de estar el modo deseado:

A = 0 → modo texto (40 columnas).
A = 1 → modo gráfico I (32 columnas).
A = 2 → modo gráfico II (alta resolución).
A = 3 → modo multicolor.

PARAMETROS DE SALIDA

A la salida la posición SCRMOD contendrá el modo en el que está trabajando el VDP.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Modifica los registros AF, BC, DE y HL. Las posibles posiciones de memoria afectadas son LINLEN, NAMBAS, CGPBAS, SCRMOD y OLDSCR.

Nota: Esta rutina llamará a las rutinas INITXT, INIT32, INIGRP o INIMULT según el valor que contenga el acumulador. A la salida se volverán a activar las interrupciones.

CHGCLR 0062

Ejecución a través de CALL &H0062.

Con esta llamada podemos cambiar los colores del texto, el papel y el fondo, si el VDP está en uno de los modos de texto (40 ó 32 columnas), o puede cambiar el color de fondo en caso que estemos en un modo gráfico.

PARAMETROS DE ENTRADA

El color del texto se toma de FORCLR; el color del papel, de BAKCLR, y el color del fondo, de BDRCLR.

PARAMETROS DE SALIDA

No devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera los registros AF, BC y HL.

CLRSPR 0069

Ejecución a través de CALL &H0069.

Esta llamada inicializa todos los *sprites*. Todos los modelos se ponen en transparente (cero), el color del *sprite* se fija al del texto, la posición vertical se pone en 209 (fuera de la pantalla) y los nombres de *sprite* se toman según el plano de *sprite* en que estén (o sea, al *sprite* que está en el plano cero se le da como nombre el código ASCII del 0, etc.).

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

No devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera el contenido de los registros AF, BC, DE y HL.

Nota: No se llama a esta rutina al cambiar de modo de pantalla.

INITXT 006C

Ejecución a través de CALL &H006C.

Esta rutina inicializa las posiciones de memoria que contienen las características de la pantalla; para el modo texto llamará a SETTXT.

PARAMETROS DE ENTRADA

Las posiciones de memoria TXTNAM, TXTCGP y LINL40 tendrán los parámetros de entrada.

PARAMETROS DE SALIDA

A la salida SCRMOD = 0, OLDSCR = 0, NAMBAS = TXTNAM, CGPBAS = TXTCGP y LINLEN = LINL40.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina modificará los contenidos de los registros AF, BC, DE y HL, además de las posiciones de memoria RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV y las posiciones de memoria indicadas en los parámetros de salida.

Nota: Esta rutina no inhibe las interrupciones.

INIT32 006F

Ejecución a través de CALL &H006F.

Esta rutina inicializa las posiciones de memoria que describen la pantalla para el modo gráfico I (32 columnas de texto); llama, por tanto, a SETT32.

PARAMETROS DE ENTRADA

Los parámetros de entrada se pasan a través de las posiciones de memoria T32NAM, T32CGP, T32COL, T32ATR, T32PAT y LINL32.

PARAMETROS DE SALIDA

A la salida tendremos SCRMOD = 1, OLDSCR = 1, NAMBAS = T32NAM, CGPBAS = CGPBAS, PATBAS = T32PAT, ATRBAS = T32 ATR y LINLEN = LINL32.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Con esta llamada se modifican los contenidos de los registros AF, BC, DE y HL, además de las posiciones de memoria RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV y las mencionadas como parámetros de salida.

Nota: Esta rutina no inhibe las interrupciones.

INIGRP 0072

Ejecución a través de CALL &H0072.

Esta rutina inicializa las posiciones de memoria para el modo gráfico II (modo de alta resolución); por tanto, llamará a SETGRP.

PARAMETROS DE ENTRADA

Las posiciones de memoria GRPNAM, GRPCGP, GRPCOL, GRPATR y GRPPAT han de contener los parámetros de entrada de esta rutina.

PARAMETROS DE SALIDA

A la salida tendremos: SCRMOD = 2, PATBAS = GRPAT y ATRBAS = GRPATR.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina alterará los contenidos de los registros AF, BC, DE y HL; las posiciones RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV y RG6SAV, y las mencionadas como parámetros de salida.

Nota: No inhibe las interrupciones.

INIMLT 0075

Ejecución a través de CALL &H0075.

Inicializa las posiciones de memoria de descripción de la pantalla para el modo multicolor; llamará a SETMLT.

PARAMETROS DE ENTRADA

Estos parámetros se pasarán a través de las direcciones MLTNAM, MLTCGP, MLTCOL, MLTATR y MLTPAT.

PARAMETROS DE SALIDA

A la salida tendremos: SCRMOD = 3, PATBAS = MLTPAT y ATRBAS = MLTATR.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina altera los contenidos de los registros AF, BC, DE y HL, además de las posiciones de memoria RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV, RG6SAV y las posiciones mencionadas como parámetros de salida.

Nota: Esta llamada no inhibe las interrupciones.

SETTXT 0078

Ejecución a través de CALL &H0078.

Esta llamada prepara los registros del procesador de video para trabajar en modo texto (40 columnas).

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

Tampoco devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Cambia el contenido de los registros AF, BC, DE y HL, y las posiciones de memoria RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV y RG6SAV.

Nota: No te basta con llamar a esta rutina para cambiar de modo de pantalla; con ella sólo se preparan los registros del VDP, pero no prepara la RAM de video. No inhibe las interrupciones.

SETT 32 007B

Ejecución a través de CALL &H007B.

Esta llamada prepara los registros del procesador de video para trabajar de modo gráfico I (32 columnas de texto).

PARAMETROS DE ENTRADA

No los necesita.

PARAMETROS DE SALIDA

No devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Altera el contenido de los registros AF, BC, DE y HL, y también los de las posiciones RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV y RG6SAV.

Nota: No es suficiente con esta llamada para cambiar a modo gráfico I de pantalla; esta rutina sólo cambia los registros de VDP; hay que inicializar también la RAM de video. No inhibe las interrupciones.

SETGRP 007E

Ejecución a través de CALL &H007E.

Esta llamada prepara los registros del procesador de video para el modo gráfico II (modo de alta resolución).

PARAMETROS DE ENTRADA

No necesita parámetro alguno.

PARAMETROS DE SALIDA

Tampoco devuelve valores.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Cambiarán los registros AF, BC, DE y HL, así como las posiciones de memoria RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV y RG6SAV.

Nota: No basta con llamar a esta rutina para cambiar de modo de pantalla. Esta rutina prepara exclusivamente los registros del VDP para el modo gráfico II, pero no prepara la RAM de video. No se inhibirán las interrupciones mientras se ejecute esta rutina.

SETMLT 0081

Ejecución a través de CALL &H0081.

Con esta llamada preparamos los registros del procesador de video para el modo multicolor.

PARAMETROS DE ENTRADA

No necesita que se le pase ningún valor.

PARAMETROS DE SALIDA

Tampoco devuelve valor alguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Modifica el contenido de los registros AF, BC, DE y HL, además de las posiciones de memoria RG0SAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV y RG6SAV.

Nota: Con llamar a esta rutina no llega para cambiar de modo. Esta rutina prepara solamente los registros del VDP, pero no cambia la RAM de video. Mientras se esté ejecutando no inhibe las interrupciones.

CALPAT 0084

Ejecución a través de CALL &H0084.

Llamando a esta rutina obtendremos la dirección de la RAM de video de un cierto modelo de *sprite*.

PARAMETROS DE ENTRADA

En el acumulador pasaremos el número del *sprite* (0-31).

PARAMETROS DE SALIDA

La dirección de la RAM de video que contiene el modelo de *sprite* la tendremos en el par de registros HL a la salida.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta rutina alterará el contenido de los registros AF, BC, DE y HL.

Nota: El interruptor de *flip-flop* no cambia su estado con esta llamada.

CALATR 0087

Ejecución a través de CALL &H0087.

Esta rutina devuelve la dirección de la RAM de video donde está la tabla de atributos de un determinado *sprite*.

PARAMETROS DE ENTRADA

En el acumulador debes pasar el número de *sprite*.

PARAMETROS DE SALIDA

A la salida tendrás la dirección de la RAM de video donde está la tabla de atributos del *sprite* especificado en el par de registros HL.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Se modificarán los contenidos de los registros DE y HL además de los *flags*.

Nota: Esta rutina no afecta a las interrupciones.

GSPSI 008A

Ejecución a través de CALL &H008A.

Esta llamada nos devuelve el tamaño, en número de bytes ocupados (8 ó 32) del *sprite* en curso.

PARAMETROS DE ENTRADA

No necesita ningún valor de entrada.

PARAMETROS DE SALIDA

A la salida tendrás el número de bytes del *sprite* en el acumulador.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo cambiarán los contenidos del acumulador y los *flags*.

Nota: Si el *sprite* es de 16 × 16, el *flag* de acarreo se pondrá a 1; de otro modo estará a 0. Esta llamada no inhibe las interrupciones.

GRPPRT 008D

Ejecución a través de CALL &H008D.

Esta llamada se emplea para escribir caracteres en la pantalla gráfica (modo de alta resolución). El carácter se escribirá en la posición en que esté el cursor gráfico.

PARAMETROS DE ENTRADA

El código del carácter a escribir debes pasarlo en el acumulador.

PARAMETROS DE SALIDA

Esta rutina no devuelve ningún valor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

No modifica ningún registro ni tampoco ninguna posición de memoria.

Nota: Esta rutina sólo se ejecutará estando en el modo gráfico II.

SCALXY 010E

Ejecución a través de CALL &H010E.

Con esta llamada se asegura que el punto pasado a través de los registros se muestra en pantalla. De no ser así se "podarán" las coordenadas del punto y se envía *(flag)* un error. La "poda" significa que si X e Y son excesivamente grandes se les asignan los valores máximos permitidos, y si son negativos se pondrán a cero. En el modo multicolor las coordenadas X e Y se dividen por 4, ya que sólo tiene 64 x 48 casillas.

PARAMETROS DE ENTRADA

Los registros BC y DE deben contener las coordenadas X e Y, respectivamente.

PARAMETROS DE SALIDA

A la salida BC y DE contendrán las coordenadas X e Y "podadas", respectivamente. Si cualquiera de las coordenadas se sale de rango el *flag* de acarreo se pondrá a 0; de no ser así estará a 1.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Cambiarán los contenidos de los registros AF, BC y DE.

Nota: Date cuenta que a las coordenadas X e Y se les aplica un factor de escala en el modo multicolor (no se lo apliques tú).

Los rangos de X e Y en los modos gráficos II o multicolor son de 0 a 255, y de 0 a 191, respectivamente.

MAPXYC 0111

Ejecución a través de CALL &H0111.

Esta rutina calcula la dirección de la RAM de video de un determinado punto, bien en modo gráfico II, bien en modo multicolor. También devuelve la posición de los bits que representan al punto dentro de dicho byte.

PARAMETROS DE ENTRADA

En el registro BC ha de estar la coordenada X, y en el registro DE la coordenada Y del punto. En el modo gráfico II, X debe estar entre 0 y 225 e Y entre 0 y 191. En el modo multicolor, X ha de estar entre 0 y 63; en cambio, Y estará entre 0 y 47. Esta rutina también emplea SCRMOD para saber el modo de

pantalla, y GRPCGP o MLTCGP para saber dónde comienzan las tablas del generador de modelos en los modos gráfico II o multicolor, respectivamente.

PARAMETROS DE ENTRADA

La dirección del punto en la RAM de video se devuelve en CLOC, y el byte que indica los bits significativos, en CMASK. En modo gráfico II el punto de interés se representa por un bit (1, color del texto; 0, color del papel). Este bit tiene la misma posición que el que está a 1 en CMASK; por ejemplo, si el punto está en el quinto bit del byte apuntado por CLOC, en CMASK habrá 0010000 en binario.

En el modo multicolor cada punto estará representado por 4 bits, los necesarios para saber su color. Del mismo modo los bits que estén a 1 en CMASK serán los que indiquen los bits de la posición apuntada por CLOC, que declaran el color del punto; por ejemplo, si el punto se representa por los bits 0-3 del byte apuntado por CLOC, en CMASK tendremos 00001111 en binario.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Se verán alterados los contenidos de los registros AF, DE y HL, así como las posiciones de memoria CLOC y CMASK.

Nota: Con esta llamada no se comprueba que el punto esté realmente representado en la pantalla. Para hacer esta comprobación y aplicar la escala a las coordenadas X e Y del punto llama a SCLXY.

FETCHC 0114

Ejecución a través de CALL &H0114.

Esta rutina lee la dirección del punto en tratamiento y los bits que lo definen.

PARAMETROS DE ENTRADA

La dirección del punto debe estar en CLOC, y en CMASK los bits que lo definen.

PARAMETROS DE SALIDA

A la salida tendrás en HL el contenido de CLOC y en el acumulador el valor almacenado en CMASK.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada modifica exclusivamente los contenidos de HL y AF.

Nota: Puedes sustituir esta llamada por estas dos instrucciones del Z80:

```
LD A,  (CMASK)
LD HL, (CLOC)
```

STOREC 0117

Ejecución a través de CALL &H0117.

Esta rutina almacena la dirección y bits significativos del punto en curso en memoria.

PARAMETROS DE ENTRADA

La dirección del punto debe estar en el registro HL, y el byte que indica los bits de definición del punto, en el acumulador.

PARAMETROS DE SALIDA

A la salida tendrás en CMASK una copia del acumulador, y en CLOC la copia de HL .

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo varían los contenidos de CMASK y CLOC.

Nota: Puedes sustituir esta llamada por estas dos instrucciones del Z80:

```
LD  (CMASK), A
LD  (CLOC),  HL
```

SETATR 011A

Ejecución a través de CALL &H011A.

Con esta llamada se asigna al byte de atributos el valor del acumulador si el contenido del acumulador es menor que 16; de no ser así, el byte de atributo no cambia y se da una señal de error.

PARAMETROS DE ENTRADA

En el acumulador has de poner el valor que quieres dar al byte de atributos.

PARAMETROS DE SALIDA

A la salida ATRBYT (byte de atributos) contendrá el valor puesto en el acumulador, si éste fuese menor que 16.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

La única posición que podrá tener cambios es ATRBYT.

Nota: El byte de atributo representa el color del punto. La rutina SETC llama a ésta para poner un punto a un color determinado.

READC 011D

Ejecución a través de CALL &H011D.

Esta rutina lee el color (atributo) del punto en curso.

PARAMETROS DE ENTRADA

En CLOC y CMASK has de poner la dirección y el byte indicador de los bits de definición del punto en curso.

PARAMETROS DE SALIDA

En la salida se almacena el color del punto en el acumulador.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo se verán alterados los contenidos del acumulador y de los *flags*.

SETC 0120

Ejecución a través de CALL &H0120.

Con esta llamada pones un color determinado a un punto. En modo multicolor sólo pondrás dicho color a ese punto. Sin embargo, en el modo gráfico II cambiarás el color de todos los puntos dibujados por el byte a que el punto pertenece, si el color dado no coincide con el de los otros puntos.

PARAMETROS DE ENTRADA

Has de poner la dirección del punto y el byte indicador de los bits de definición en las posiciones CLOC y CMASK, respectivamente. En ATRBYT estará el color en que queremos poner el punto.

PARAMETROS DE SALIDA

No devuelve parámetro alguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo se verán alterados los contenidos del acumulador y los *flags*.

Nota: Si todos los puntos que indica un byte se ponen el el mismo color, éste se convertirá en el color de fondo; o sea, el byte que apunta a CLOC de la RAM de video será cero.

RIGHTC 00FC

Ejecución a través de CALL &H00FC.

Esta rutina mueve el cursor gráfico un punto a la derecha.

PARAMETROS DE ENTRADA

La dirección del punto actual debe estar en CLOC, y el byte de definición, en CMASK.

PARAMETROS DE SALIDA

Los contenidos de CLOC y CMASK se actualizarán según la nueva posición del cursor gráfico.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Esta llamada altera los contenidos del registro AF y de las posiciones CLOC y CMASK.

Nota: Si al mover el cursor a la derecha se sale de la pantalla, aparecerá de nuevo por el lado izquierdo, pero un punto más abajo; no se hace ningún test para comprobar que el cursor se ha salido de la pantalla a través del punto más inferior a la derecha.

En el modo multicolor el cursor se mueve de cuatro en cuatro puntos.

LEFTC 00FF

Ejecución a través de CALL &H00FF.

Esta rutina desplaza el cursor gráfico un punto a la izquierda.

PARAMETROS DE ENTRADA

En CLOC debe estar la dirección del byte que contiene al punto actual; en CMASK tendrás el byte de definición de dicho punto.

PARAMETROS DE SALIDA

Se actualizarán los contenidos de CLOC y CMASK según la posición del cursor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Se alteran los contenidos de AF, CLOC y CMASK.

Nota: Si el cursor se sale por la parte izquierda de la pantalla, aparecerá por la derecha, pero un punto más arriba; en este caso, tampoco se comprueba que se se salga por completo de la pantalla a través del punto situado más arriba a la izquierda.

En modo multicolor el cursor se mueve de cuatro en cuatro puntos.

UPC 0102

Ejecución a través de CALL &H0102.

Esta rutina mueve un punto hacia arriba el cursor gráfico. Si se encuentra en el tope superior, no hará nada.

PARAMETROS DE ENTRADA

En CLOC ha de estar la dirección del byte que contiene al punto en curso, y en CMASK estará el byte que indica los bits de definición del punto.

PARAMETROS DE SALIDA

El contenido de las posiciones CLOC y CMASK se actualizará según la nueva posición del cursor gráfico.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Se verán alterados los contenidos del registro AF y las posiciones CLOC y CMASK.

Nota: En el modo multicolor el cursor se moverá en bloques de 4 × 4 puntos.

TUPC 0105

Ejecución a través de CALL &H0105.

Esta rutina mueve el cursor un punto hacia arriba, en caso de que éste no esté en el tope; de ser así se mantiene la posición del cursor y se pone a 1 el *flag* de acarreo.

PARAMETROS DE ENTRADA

En CLOC y CMASK estarán la dirección del byte que contiene al punto en curso y el byte indicador de los bits de declaración del punto, respectivamente.

PARAMETROS DE SALIDA

Los contenidos de CLOC y CMASK se actualizarán según la nueva posición del cursor gráfico.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo variarán los contenidos del registro A y las posiciones CLOC y CMASK.

Nota: Esta rutina es prácticamente igual a la UPC, con la diferencia que ésta pone a 1 el *flag* de acarreo si se encuentra en la línea superior, y a 0 cuando está en cualquier otra.

En el modo multicolor el cursor se mueve en bloques de 4 × 4 puntos.

DOWNC 0108

Ejecución a través de CALL &H0108.

Esta rutina hace que el cursor baje una línea; en caso de que se encuentre en la línea inferior, no hará nada.

PARAMETROS DE ENTRADA

La dirección del punto actual debe almacenarse en CLOC, y en CMASK tendremos el byte que indican los bits de definición del punto.

PARAMETROS DE SALIDA

Los contenidos de CLOC y CMASK se actualizarán según la posición en que se encuentre el cursor gráfico.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo cambiarán los contenidos del registro AF y las posiciones CLOC y CMASK.

Nota: En el modo multicolor el cursor se mueve en bloques de 4 × 4 puntos.

TDOWNC 010B

Ejecución a través de CALL &H010B.

Esta rutina mueve el cursor gráfico un punto hacia abajo; en caso de estar en la línea inferior no se moverá y pondrá a 1 el *flag* de acarreo.

PARAMETROS DE ENTRADA

CLOC ha de contener la dirección del byte del punto actual; en CMASK debes tener la descripción de los bits de definición de dicho punto.

PARAMETROS DE SALIDA

A la salida se actualizarán los contenidos de CLOC y CMASK, según la nueva posición del cursor.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Se verán alterados los contenidos de CLOC y CMASK, así como el registro AF.

Nota: Esta llamada es prácticamente igual que DOWNC, pero con la diferencia de que ésta pone a 1 el *flag* de acarreo si el cursor gráfico está en la línea más inferior, y a 0 si está en cualquier otra línea.

En el modo multicolor el cursor se mueve de cuatro en cuatro puntos.

NSETCX 0123

Ejecución a través de CALL &H0123.

Esta llamada pone a un determinado color un cierto número de puntos situados a la derecha del cursor gráfico.

PARAMETROS DE ENTRADA

En HL debes tener el número de puntos a ubicar; en ATRBYT, el color, y en CLOC y CMASK, la dirección y byte de indicación del punto donde está situado el cursor, respectivamente.

PARAMETROS DE SALIDA

No devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Se verán alterados los contenidos de AF, BC, DE, HL y las posiciones de memoria CLOC y CMASK.

Nota: En modo multicolor, el cursor se moverá un punto a la izquierda; en cambio, en el modo gráfico 2 no se moverá.

Esta rutina no inhibe las interrupciones.

GTASPC 0126

Ejecución a través de CALL &H0126.

El intérprete de BASIC emplea esta rutina para la ejecución de la instrucción CIRCLE; obtiene los parámetros de variación del radio.

PARAMETROS DE ENTRADA

La posición ASPECT1 contendrá 256/(variación del radio) y ASPECT2 256*(variación del radio).

PARAMETROS DE SALIDA

A la salida tendremos en HL el contenido de ASPECT2, y en DE el contenido de ASPECT1.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo cambiarán los contenidos de HL y DE.

PNTINI 0129

Ejecución a través de CALL &H0129.

Es una preparación para la instrucción PAINT del BASIC. Analiza el color del borde de la figura y, si es menor que 16, lo almacena.

PARAMETROS DE ENTRADA

En el acumulador debe almacenarse el color del borde de la figura a colorear.

PARAMETROS DE SALIDA

Si estás en el modo multicolor, BRDATR tendrá una copia del parámetro de entrada; de no estar en ese modo dicha posición tendrá una copia de ATRBYT. Si el parámetro de entrada es mayor que 15 y estás en modo multicolor, el bit de acarreo se pondrá a 1 a la salida.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Sólo se verán afectados los contenidos de la posición BRDATR, del acumulador y de los *flags*.

Nota: El color del borde sólo se emplea con la instrucción PAINT estando en el modo multicolor. En el modo gráfico II la instrucción PAINT sólo emplea el color de fondo.

SCANR 012C

Ejecución a través de CALL &H012C.

Con esta llamada se avanzan puntos desde la derecha del cursor gráfico, poniéndolos a la vez de un determinado color. El avance cesa cuando encontramos un punto con un cierto color (de borde) o llegamos al tope derecho de la pantalla, o bien se haya avanzado el máximo número de puntos.

PARAMETROS DE ENTRADA

En DE has de poner el número máximo de puntos (hasta 256); BRDATR tendrá el color del borde, y en ATRBYT estará el color que quieres dar a los puntos avanzados.

PARAMETROS DE SALIDA

No devuelve parámetro alguno.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

AF, BC, DE y HL son los registros cuyo contenido se alterará. Las posiciones CLOC, CMASK y de FILNAM a FILNAM+3 también se verán afectados.

Nota: La instrucción PAINT se apoya en esta rutina. Las posiciones de FILNAM a FILNAM+3 se emplean como área de trabajo.

Esta rutina no inhibe las interrupciones.

SCANL 012F

Ejecución a través de CALL &H012F.

Esta subrutina avanza un determinado número de puntos a la izquierda del cursor, poniéndolos de un determinado color; el avance cesará al encontrar un punto con un determinado color de borde fijado o al llegar al tope izquierdo de la pantalla o al haber recorrido un número máximo de puntos.

PARAMETROS DE ENTRADA

El número máximo de puntos a recorrer estará en el registro DE (hasta 256); el color del borde lo fijas en BRDATR, y el color de los puntos estará en ATRBYT.

PARAMETROS DE SALIDA

No devuelve ningún parámetro.

REGISTROS Y POSICIONES DE MEMORIA AFECTADOS

Cambiará el contenido de los registros AF, BC, DE y HL, así como las posiciones CLOC, CMASK y de FILNAM a FILNAM+3.

Nota: La instrucción PAINT del BASIC emplea esta rutina. Las posiciones que van de FILNAM a FILNAM+3 se utilizan como área de trabajo.

Con esta llamada no se inhiben las interrupciones.

60

Los vectores

Los vectores nos facilitan la tarea de interceptar al intérprete BASIC o al sistema operativo en determinados puntos para ejecutar procesos adicionales o alternativos. Veámoslo con un ejemplo. Toma la BIOS CHPUT (véase el tema 55). Esta rutina se emplea para escribir un texto en pantalla. Estas son las primeras instrucciones de CHPUT escritas en ensamblador:

```
PUSH   HL          APILA EL CONTENIDO DE TODOS LOS
                   REGISTROS.
PUSH   DE
PUSH   BC
PUSH   AF
CALL   &HFDA4      ; LLAMA AL VECTOR H.CHPU.
```

Primero se apilan los registros HL, DE, BC y AF. Entonces se hace una llamada a la dirección &HFDA4. Generalmente esta posición y los cuatro bytes siguientes contienen la instrucción RET del Z80; por tanto, se cede el control a la siguiente instrucción a la llamada. Pero al estar esta dirección en la RAM, podemos cambiar su contenido. Cambia el contenido de la posición &HFDA4 y los dos bytes siguientes para que se ceda el control a otra subrutina (&D000). O sea:

```
FDA4  CE 00 D0     JP      &HD000
```

Mételo aquí codificado a partir de la posición &HD000:

```
POP     HL      ; DESAPILA LA CIMA DE LA PILA (VUELVE
                  A LA DIRECCION DEL VECTOR).

POP     AF      ; TOMA EL CONTENIDO DE LOS REGIS-
                  TROS.
POP     BC
POP     DE
POP     HL
RET             ; SALIDA DE LA BIOS CHPUT.
```

El dato de la cima de la pila, que ahora es la dirección de retorno a CHPUT, es tres unidades mayor que la dirección donde se ubica la llamada a &HFDA4. Si lo quitamos, y además desapilamos los contenidos de los registros, tendremos en la cima de la pila la dirección de retorno de la llamada a CHPUT; por tanto, al hacer RET volveremos a la instrucción siguiente a la llamada de la subrutina CHPUT. De esta forma no se escribirá en pantalla el dato.

En este ejemplo te demostramos cómo se puede interceptar al sistema operativo. Los cinco bytes a partir de la dirección &HFDA4 se denominan "vector".

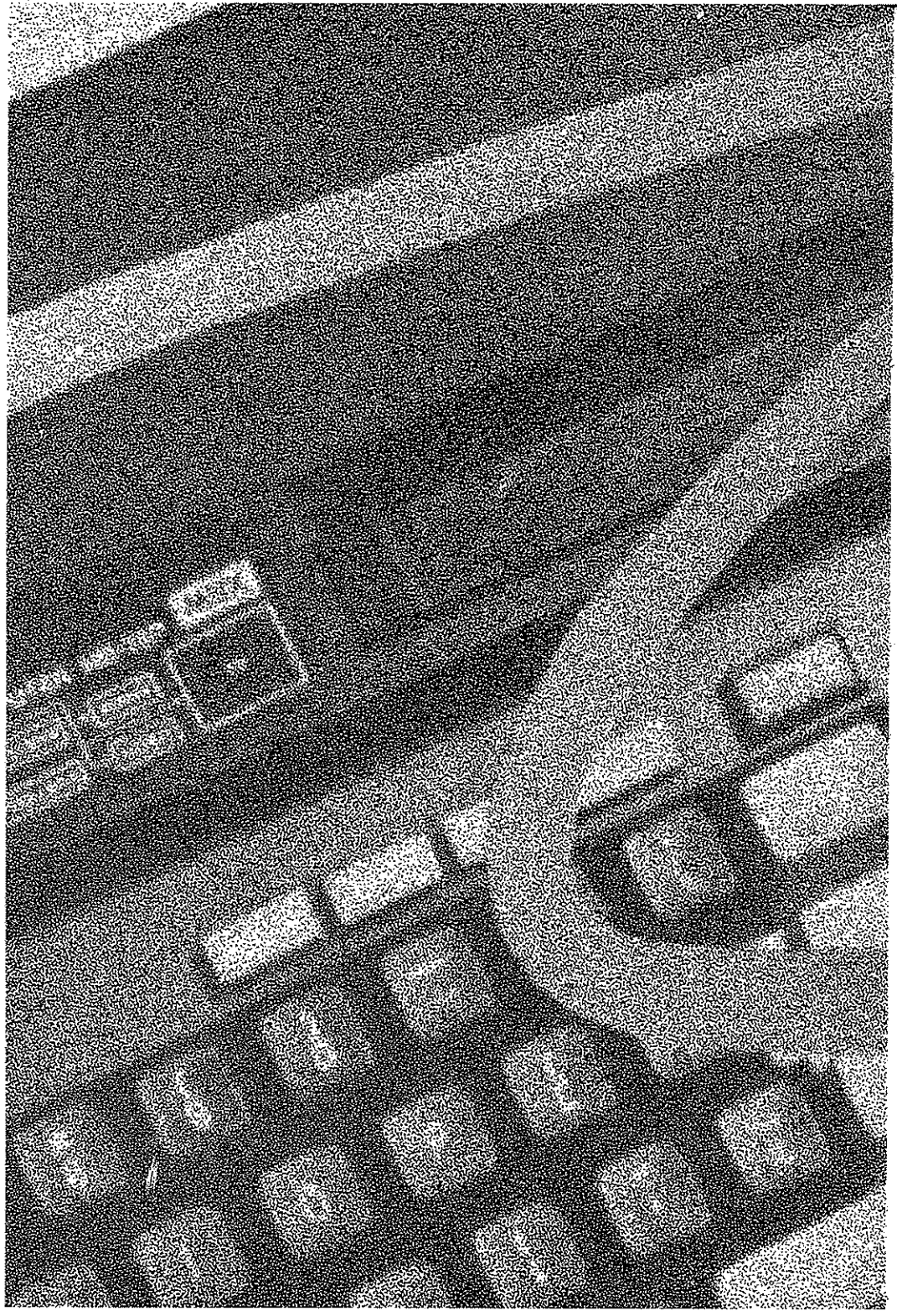
El sistema operativo y el intérprete de BASIC tienen muchos otros vectores. (A continuación tienes una lista con todos ellos en orden alfabético.)

NOMBRE	DIRECCION	EXPLICACION
H-ATTR	FE1C	MSXSTS, al comienzo de la rutina de atributos, ATTR$.
H-BAKU	FEAD	SPCDSK, en la rutina de vuelta atrás, BAKUPT.
H-BINL	FE76	SPCDSK, la rutina de carga binaria, BINLOD.
H-BINS	FE71	SPCDSK, en la rutina de grabación binaria, BINSAV.
H-BUFL	FF8E	BINTRP, en la rutina del *buffer* de línea, BUFLIN.
H-CHGE	FDC2	MSXIO, al comienzo de la rutina de lectura de carácter, CHGET.
H-CHPU	FDA4	MSXIO, al comienzo de la rutina de escritura de carácter, CHPUT.
H-CHRG	FF48	BINTRP.
H-CLEA	FED0	BIMISC, en la rutina de borrado, CLEARC.
H-CMD	FE0D	MSXSTS, al comienzo de la rutina de comando, CMD.
H-COMP	FF57	BINTRP.
H-COPY	FE08	MSXSTS, al comienzo de la rutina de copia de ficheros, COPY.
H-CRDO	FEE9	BIO, en la rutina CRDO.
H-CRUN	FF20	BINTRP.
H-CRUS	FF25	BINTRP.
H-CVD	FE49	MSXSTS, al comienzo de la rutina de conversión a doble precisión, CVD.
H-CVI	FE3F	MSXSTS, al comienzo de la rutina de conversión a entero, CVI.
H-CVS	FE44	MSXSTS, al comienzo de la rutina de conversión a precisión simple, CVS.
H-DEVN	FEC1	SPCDEV, en la rutina de nombre de unidad, DEVNAM.
H-DGET	FE80	SPCDSK, en la rutina de lectura de disco, DGET.
H-DIRD	FF11	BINTRP, en la rutina de ejecución directa de instrucción, DIRDO.

NOMBRE	DIRECCION	EXPLICACION
H-DOGR	FEF3	GENGRP, en la rutina de gráficos, DOGRPH.
H-DSKC	FEEE	BIO, en la rutina de lectura de un carácter de disco, DSKCHI.
H-DSKF	FE12	MSXSTS, al comienzo de la rutina de disco libre, DSKF.
H-DSKI	FE17	MSXSTS, al comienzo de la rutina de escritura en disco, DSKI.
H-DSKO	FDEF	MSXSTS, al comienzo de la rutina de lectura de disco, DSKO$.
H-DSPC	FDA9	MSXIO, al comienzo de la rutina de visualizado del cursor, DSPCSR.
H-DSPF	FDB3	MSXIO, al comienzo de la rutina de visualización del contenido de las teclas de función, DSPFNK.
H-EOF	FEA3	SPCDSK, en la función de fin de fichero, EOF.
H-ERAC	FDAE	MSXIO, al comienzo de la rutina de borrado del cursor, ERACSR.
H-ERAF	FDB8	MSXIO, al comienzo de la rutina de borrado del contenido de las teclas de función ERAFNK.
H-ERRF	FF02	BINTRP.
H-ERRO	FFB1	BINTRP, en la rutina de error, ERROR.
H-ERRP	FEFD	BINTRP, en la rutina de escritura de mensaje de error, ERRPRT.
H-EVAL	FF70	BINTRP.
H-FIEL	FE2B	MSXSTS, al comienzo de la rutina de campo, FIELD.
H-FILE	FE7B	SPCDSK, en el comando FILES.
H-FILO	FE85	SPCDSK, al comienzo de la rutina FILOU1 (escritura en disco 1).
H-FINE	FF1B	BINTRP.
H-FING	FF7A	BINTRP.
H-FINI	FF16	BINTRP.
H-FINP	FF5C	BINTRP.
H-FORM	FFAC	MSXIO, en la rutina formateadora de disco, FORMAT.
H-FPOS	FEA8	SPCDSK, en la función de posición en fichero FPOS.
H-FRET	FF9D	BISTRG, en la rutina de espacio libre temporal, FRETM.
H-FRME	FF66	BINTRP.
H-FRQI	FF93	BINTRP, en la rutina FRQINT.
H-GEND	FEC6	SPCDEV, en la rutina general de reparto de unidades, GENDSP.
H-GETP	FE4E	SPCDSK, en la rutina de lectura de puntero de fichero, GETPTR.
H-GONE	FF43	BINTRP.
H-INDS	FE8A	SPCDSK, en la rutina de lectura de un carácter del disco, INDSKC.
H-INIP	FDC7	MSXIO, al comienzo de la rutina de inicialización de modelo, INIPAT.
H-INLI	FDE5	MSXINL, al comienzo de la rutina de lectura de línea, LININ.
H-IPL	FE03	MSXSTS, al comienzo de la rutina de carga inicial de programa, IPL.
H-ISFL	FEDF	BIMISC, al comienzo de la rutina de análisis de E/S de fichero, ISFLIO.
H-ISMI	FF7F	BINTRP, en la rutina de análisis de MID$, ISMID$.
H-ISRE	FF2A	BINTRP.
H-KEYC	FDCC	MSXIO, al comienzo de la rutina de codificación de clave, KEYCOD.
H-KEYI	FD9A	MSXIO, al comienzo del tratamiento de interrupciones.
H-KILL	FDFE	MSXSTS, al comienzo de la rutina de destrucción de fichero, KILL.
H-KYEA	FDD1	MSXIO, al comienzo de la rutina de clave simple, KYEASY.

NOMBRE	DIREC-CION	EXPLICACION
H-LIST	FF89	BINTRP, en la rutina de LIST.
H-LOC	FE99	SPCDSK, en la función de posición, LOC.
H-LOF	FE9E	SPCDSK, en la función de longitud de fichero, LOF.
H-LOPD	FED5	BIMISC, en la rutina de bucle e inicialización estándar, LOPDFT.
H-LPTO	FFB6	MSXIO, en la rutina de escritura de línea en impresora, LPTOUT.
H-LPTS	FFBB	MSXIO, en la rutina de estado de línea de impresora, LPTSTT.
H-LSET	FE21	MSXSTS, al comienzo de la rutina de inicialización de izquierda, LSET.
H-MAIN	FF0C	BINTRP, en la entrada principal, MAIN.
H-MERG	FE67	SPCDSK, en la rutina de unión de ficheros con programas, MERGE.
H-MKD$	FE3A	MSXSTS, en la rutina de creación de doble precisión, MKD$.
H-MKI$	FE30	MSXSTS, al comienzo de la rutina de creación de entero, MKI$.
H-MKS$	FE35	MSXSTS, al comienzo de la rutina de creación de precisión, simple, MKS$.
H-NAME	FDF9	MSXSTS, al comienzo de la rutina de renombrar, NAME.
H-NEWS	FF3E	BINTRP.
H-NMI	FDD6	MSXIO, al comienzo de la rutina de interrupción no enmascarable, NMI.
H-NODE	FEB7	SPCDEV, en la rutina de falta de nombre de unidad NODEVN.
H-NOFO	FE58	SPCDSK, en la rutina de falta de cláusula for, NOFOR.
H-NOTR	FF34	BINTRP.
H-NTFL	FE62	SPCDSK, en la rutina 0 de falta de número de fichero, NTFLO.
H-NTFN	FF2F	BINTRP.
H-NTPL	FF6B	BINTRP.
H-NULO	FE5D	SPCDSK, en la rutina de apertura de fichero nulo, NULOPN.
H-OKNO	FF75	BINTRP.
H-ONGO	FDEA	MSXSTS, al comienzo de la rutina en goto procedimiento, ONGOTP.
H-OUTD	FEE4	BIO, en la rutina de salida, OUTDO.
H-PARB	FEB2	SPCDEV, en la rutina de petición de nombre de unidad, PARDEV.
H-PHYD	FFA7	MSXIO, en la rutina de *diskette* real, PHYDIO.
H-PINL	FDDB	MSXINL, al comienzo de la rutina de línea del programa, PINLIN.
H-PLAY	FFC5	MSXSTS, en la entrada de la instrucción PLAY.
H-POSD	FEBC	SPCDEV, en la rutina de disco accesible, POSDSK.
H-PRGE	FEF8	BINTRP, en la rutina de fin de programa, PRGEND.
H-PRTF	FF52	BINTRP.
H-PTRG	FFA2	BIPTRG, en la rutina de lectura de puntero, PTRGET.
H-QINL	FDE0	MSXINL; al comienzo de la rutina de interrogación y entrada de línea, QINLIN.
H-READ	FF07	BINTRP, en entrada preparada.
H-RETU	FF4D	BINTRP.
H-RSET	FE26	MSXSTS, al comienzo de la rutina de la inicialización derecha, RSET.
H-RSLF	FE8F	SPCDSK, para reseleccionar la antigua unidad.
H-RUNC	FECB	BIMISC, en la rutina de borrado de ejecución, RUNC.
H-SAVD	FE94	SPCDSK, para grabar en la unidad actual.
H-SAVE	FE6C	SPCDSK, en la rutina de SAVE.
H-SCNE	FF98	BINTRP.

NOMBRE	DIREC-CION	EXPLICACION
H-SCRE	FFC0	MSXSTS, en la entrada de la instrucción SCREEN.
H-SETF	FE53	SPCDSK, en la rutina de inicialización del puntero de fichero, SETFIL.
H-SETS	FDF4	MSXSTS, al comienzo de la rutina de inicialización de atributos.
H-SNGF	FF39	BINTRP.
H-STKE	FEDA	BIMISC, en la rutina de error en la pila, STKERR.
H-TIMI	FD9F	MSXIO, al comienzo del tratamiento de interrupciones.
H-TOTE	FDBD	MSXIO, al comienzo de la rutina de paso forzado a pantalla en modo texto, TOTEXT.
H-TRMN	FF61	BINTRP.
H-WIDT	FF84	BINTRP, en la rutina de anchura, WIDTHS.



61
La RAM del sistema

En este tema te enumeramos todas las variables del sistema con su dirección y su longitud en bytes. La dirección viene dada en hexadecimal, y la longitud, en decimal. La RAM del sistema va desde la posición &HF380 a la posición &HFFFF. De la dirección &HF380, inclusive, hay rutinas en código máquina, que son llamadas por las BIOS de tratamiento de cartucho.

NOMBRE	DIRECCION	LONGITUD	NOMBRE	DIRECCION	LONGITUD
ARG	F847	16	BUFEND	FC18	0
ARYTA2	F7B5	2	BUFMIN	F55D	1
ARYTAB	F6C4	2	CAPST	FCAB	1
ASPCT1	F40B	2	CASPRV	FCB1	1
ASPCT2	F40D	2	CENCNT	F933	2
ASPECT	F931	2	CGPBAS	F924	2
ATRBAS	F928	2	CGPNT	F91F	3
ATRBYT	F3F2	1	CLIKFL	FBD9	1
AUTFLG	F6AA	1	CLIKSW	F3DB	1
AUTINC	F6AD	2	CLINEF	F935	1
AUTLIN	F6AB	2	CLMLST	F3B2	1
BAKCLR	F3EA	1	CLOC	F92A	2
BASROM	FBB1	1	CLPRIM	F38C	14
BDRCLR	F3EB	1	CMASK	F92C	1
BOTTOM	FC48	2	CNPNTS	F936	2
BRDATR	FCB2	1	CNSDFG	F3DE	1
BUF	F55E	258	CODSAV	FBCC	1

NOMBRE	DIRECCION	LONGITUD	NOMBRE	DIRECCION	LONGITUD
CONLO	F66A	8	FNKSWI	FBCD	1
CONSAV	F668	1	FORCLR	F3E9	1
CONTXT	F666	2	FRCNEW	F3F5	1
CONTYP	F669	1	FRETOP	F69B	2
CPCNT	F939	2	FSTPOS	FBCA	2
CPCNT8	F93B	2	FUNACT	F7BA	2
CPLOTF	F938	1	GETPNT	F3FA	2
CRCSUM	F93D	2	GRPACX	FCB7	2
CRTCNT	F3B1	1	GRPACY	FCB9	2
CS120	F3FC	10	GRPATR	F3CD	2
CSAVEA	F942	2	GRPCGP	F3CB	2
CSAVEM	F944	1	GRPCOL	F3C9	2
CSCLXY	F941	1	GRPHED	FCA6	1
CSRSW	FCA9	1	GRPNAM	F3C7	2
CSRX	F3DD	1	GRPPAT	F3CF	2
CSRY	F3DC	1	GXPOS	FCB3	2
CSTCNT	F93F	2	GYPOS	FCB5	2
CSTYLE	FCAA	1	HEADER	F40A	1
CURLIN	F41C	2	HIGH	F408	2
CXOFF	F945	2	HIMEM	FC4A	2
CYOFF	F947	2	HOLD	F83E	8
DAC	F7F6	16	HOLD2	F836	8
DATLIN	F6A3	2	HOLD8	F806	48
DATPTR	F6C8	2	INSFLG	FCA8	1
DECCNT	F7F4	1	INTCNT	FCA2	2
DECTM2	F7F2	2	INTFLG	FC9B	1
DECTMP	F7F0	2	INTVAL	FCA0	2
DEFTBL	F6CA	26	JIFFY	FC9E	2
DEVICE	FD99	1	KANAMD	FCAD	1
DIMFLG	F662	1	KANAST	FCAC	1
DONUM	F665	1	KBUF	F41F	318
DORES	F664	1	KEYBUF	FBF0	40
DOT	F6B5	2	LFPROG	F954	1
DRWANG	FCBD	1	LINL32	F3AF	1
DRWFLG	FCBB	1	LINL40	F3AE	1
DRWSCL	FCBC	1	LINLEN	F3B0	1
DSCTMP	F698	3	LINTTB	FBB2	24
ENDBUF	F660	1	LINWRK	FC18	40
ENDFOR	F6A1	2	LOHADR	F94B	2
ENDPRG	F40F	5	LOHCNT	F94D	2
ENSTOP	FBB0	1	LOHDIR	F94A	1
ERRFLG	F414	1	LOHMSK	F949	1
ERRLIN	F6B3	2	LOW	F406	2
ERRTXT	F6B7	2	LOWLIM	FCA4	1
ESCCNT	FCA7	1	LPTPOS	F415	1
EXPTBL	FCC1	4	MAXDEL	F92F	2
FBUFFR	F7C5	43	MAXFIL	F85F	1
FILNAM	F866	11	MAXUPD	F3EC	3
FILNM2	F871	11	MCLFLG	F958	1
FILTAB	F860	2	MCLLEN	FB3B	1
FLBMEM	FCAE	1	MCLPTR	FB3C	2
FLGINP	F6A6	1	MCLTAB	F956	2
FNKFLG	FBCE	10	MEMSIZ	F672	2
FNKSTR	F87F	160	MINDEL	F92D	2

NOMBRE	DIRECCION	LONGITUD	NOMBRE	DIRECCION	LONGITUD
MINUPD	F3EF	3	RG6SAV	F3E5	1
MLTATR	F3D7	2	RG7SAV	F3E6	1
MLTCGP	F3D5	2	RNDX	F857	8
MLTCOL	F3D3	2	RS2IQ	FAF5	64
MLTNAM	F3D1	2	RTPROG	F955	1
MLTPAT	F3D9	2	RTYCNT	FC9A	1
MOVCNT	F951	2	RUNBNF	FCBE	1
MUSICF	FB3F	1	RUNFLG	F866	0
NAMBAS	F922	2	SAVEND	F87D	2
NEWKEY	FBE5	11	SAVENT	FCBF	2
NLONLY	F87C	1	SAVSP	FB36	2
NOFUNS	F7B7	1	SAVSTK	F6B1	2
NTMSXP	F417	1	SAVTXT	F6AF	2
NULBUF	F862	2	SAVVOL	FB39	2
OLDKEY	FBDA	11	SCNCNT	F3F6	1
OLDLIN	F6BE	2	SCRMOD	FCAF	1
OLDSCR	FCB0	1	SKPCNT	F94F	2
OLDTXT	F6C0	2	SLTATR	FCC9	64
ONEFLG	F6BB	1	SLTTBL	FCC5	4
ONELIN	F6B9	2	SLTWRK	FD09	128
ONGSBF	FBD8	1	STATFL	F3E7	1
OPRTYP	F664	0	STKTOP	F674	2
PADX	FC9D	1	STREND	F6C6	2
PADY	FC9C	1	SUBFLG	F6A5	1
PARM1	F6E8	100	SWPTMP	F7BC	8
PARM2	F750	100	T32ATR	F3C3	2
PATBAS	F926	2	T32CGP	F3C1	2
PATWRK	FC40	8	T32COL	F3BF	2
PDIREC	F953	1	T32NAM	F3BD	2
PLYCNT	FB40	1	T32PAT	F3C5	2
PRMFLG	F7B4	1	TEMP	F6A7	2
PRMLEN	F6E6	2	TEMP2	F6BC	2
PRMLN2	F74E	2	TEMP3	F69D	2
PRMPRV	F74C	2	TEMP8	F69F	2
PRMSTK	F6E4	2	TEMP9	F7B8	2
PROCNM	FD89	16	TEMPPT	F678	2
PRSCNT	FB35	1	TEMPST	F67A	30
PRTFLG	F416	1	TRCFLG	F7C4	1
PTRFIL	F864	2	TRGFLG	F3E8	1
PTRFLG	F6A9	1	TRPTBL	FC4C	78
PUTPNT	F3F8	2	TTYPOS	F661	1
QUEBAK	F971	4	TXTATR	F3B9	2
QUETAB	F959	24	TXTCGP	F3B7	2
QUEUEN	FB3E	1	TXTCOL	F3B5	2
QUEUES	F3F3	2	TXTNAM	F3B3	2
RAWPRT	F418	1	TXTPAT	F3BB	2
RDPRIM	F380	5	TXTTAB	F676	2
REPCNT	F3F7	1	USFLG	F6A6	0
RG0SAV	F3DF	1	USRTAB	F39A	20
RG1SAV	F3E0	1	VALTYP	F663	1
RG2SAV	F3E1	1	VARTAB	F6C2	2
RG3SAV	F3E2	1	VCBA	FB41	37
RG4SAV	F3E3	1	VCBB	FB66	37
RG5SAV	F3E4	1	VCBC	FB8B	37

NOMBRE	DIRECCION	LONGITUD	NOMBRE	DIRECCION	LONGITUD
VLZADR	F419	2	VOICCQ	FA75	128
VLZDAT	F41B	1	VOICEN	FB38	1
VOICAQ	F975	128	WINWID	FCA5	1
VOICBQ	F9F5	128	WRPRIM	F385	7

Indice alfabético

Los números de página en negrita hacen referencia a la parte tercera: "Instrucciones del BASIC".

MSX-GUIA DEL PROGRAMADOR Y MANUAL DE REFERENCIA es el libro definitivo sobre el *standard* MSX, imprescindible para cualquier usuario, experto o novato, de un microordenador MSX.

Cubre todo lo que se necesita saber sobre MSX, desde el BASIC al Sistema Operativo, de forma exhaustiva pero con un estilo conciso, estando eficazmente estructurado para que se pueda utilizar de forma práctica como manual de consulta y referencia constantes o para aprender a programar y manejar con destreza cualquier micro MSX.

El libro está organizado en cuatro partes que facilitan el acceso rápido a la información que se desee consultar:

La primera incluye una completa introducción al BASIC MSX y a los macrolenguajes gráfico y musical.

La segunda es una guía avanzada de programación que incluye gráficos avanzados, acceso al TMS 9929A (VDP) y al AY-3-8912 (PSG), manejo de ficheros, ejecución de rutinas en código máquina y programación con periféricos.

La tercera constituye una completa guía de consulta y referencia para el programador en BASIC o código máquina.

Al final del libro se describe detalladamente el sistema operativo MSX: instrucciones RST, puntos de entrada y salida, BIOS y vectores.

Cualquiera que sea el micro MSX que se utilice, MSX-GUIA DEL PROGRAMADOR Y MANUAL DE REFERENCIA es la obra imprescindible en la biblioteca de cualquier programador.