

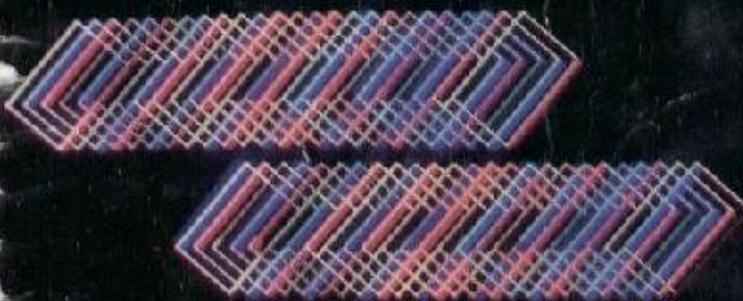
SONY®

MSX

date food elec edu

date	food	elec	edu
May 1	4550		98
May 2	3500		
May 3	6800		750
May 4	1100	850	1900
May 5	4000		
May 6	8000		

# Aprenda a programar en BASIC-MSX



DEMONSTRATION

FIG. 1 BAR GRAPH

FIG. 2 LINE GRAPH

FIG. 3 CIRCLE GRAPH

```

    DE-17
    B=20
    C=00
    C=C-1
    C<0 ?
    B=B-1
    B=0 ?
    PCT
  
```

HIT BIT

CON PROGRAMAS



# APRENDA A PROGRAMAR EM BASIC MSX

Copyright © 1987 by Editora de Informática  
Livraria da Informática - São Paulo, SP  
ISBN 85-709-4411-2  
CDD 780.13  
Todos os direitos reservados de ARSIL CORP.  
1987

3ª Edición

© 1987 SONY ESPAÑA, S.A.

Sabino de Arana 42-44

08028-Barcelona

Reservados todos los derechos. Ninguna parte de este libro puede ser reproducida por ningún medio, sin el permiso previo de SONY ESPAÑA, S.A.

Dpto. Legal B: 3940-87

ISBN 84-398-4421-2

Imprime: Gráficas Signo, S.A.

Carretera de Cornellá, 140, 2º — Esplugues de Llobregat

*Printed in Spain.* Impreso en España.

MSX es marca registrada de ASCII CORP.

COD. 21000195

## PROLOGO

Hasta la aparición del MSX, los ordenadores existentes en el mercado se caracterizaban por su incompatibilidad. Los programas y los distintos periféricos conectables a la unidad central eran solamente válidos para aquella marca que estaban previstos.

Esta incompatibilidad era, y es, un grave inconveniente para el usuario final, que queda indefenso ante los vaivenes que pueda sufrir cada una de estas empresas.

El MSX ha revolucionado el mercado en este aspecto. Son ya más de veinte compañías que han adoptado este standard, dispuestas a terminar con la anarquía de lenguajes y de conexión de periféricos existentes.

Se pretende así ofrecer una amplia gama de programas, ordenadores y periféricos para que el usuario pueda escoger aquello que mejor se adapte a sus necesidades de entre todos los modelos existentes, contando con la seguridad de que podrá adquirir programas y periféricos de cualquiera de las compañías MSX.

Pero esta es solamente una de las ventajas del MSX.

La segunda, tan importante o más que la primera, es su propia potencia. A lo largo de las páginas de este libro se comentan todos los comandos del MSX-BASIC para que lo pueda comprobar.

Se ofrecen, además, una serie de informaciones técnicas de gran utilidad para todos aquellos que pretendan aprender programación y también para quienes ya desean profundizar en el MSX.

Se listan, asimismo, una serie de programas con el fin de demostrar la sencillez y, al propio tiempo, la potencia del MSX-BASIC.

Podemos concluir diciendo que las tres características que configuran el MSX son su estandarización de software y hardware, su sencillez y su potencia.

Por ello puede decirse que el MSX está revolucionando la informática familiar y abre un nuevo campo en la informática aplicada a la educación y a la gestión, permitiendo disponer de un equipo completo y potente a un precio muy asequible.

PROLOGO

Desde la aparición del MSX, los ordenadores existentes en el mundo...

Esta incomprensión era y es un grave inconveniente para el...

El MSX ha revolucionado el mundo de los ordenadores...

Se pretende así ofrecer una información completa...

Pero esta es solamente una de las ventajas del MSX...

La segunda gran ventaja es que el MSX...

En tercer lugar, una serie de mejoras...

De igual manera, el MSX...

Por eso, cuando se dice que el MSX...

Por ello, puede decirse que el MSX...

# INDICE:

## PRIMERA PARTE

### INFORMACION GENERAL DEL BASIC MSX

1.0. INTRODUCCION .....	13
1.1. MODOS DE FUNCIONAMIENTO .....	13
1.2. FORMATO DE LAS LINEAS .....	13
1.2.1. Número de línea .....	14
1.3. CONJUNTO DE CARACTERES .....	14
1.4. CONSTANTES .....	15
1.4.1. Constantes numéricas de simple y doble precisión .....	16
1.5. VARIABLES .....	17
1.5.1 Nombres de variables y Declaración de caracteres .....	17
1.5.2. Variables matriciales (Variables subindicadas) .....	18
1.5.3. Requisitos de espacio .....	19
1.6. CONVERSION DE TIPOS .....	19
1.7. EXPRESIONES Y OPERADORES .....	21
1.7.1. Operadores aritméticos .....	21
1.7.1.1. División entera y Resto .....	21
1.7.1.2. Overflow y Division por cero .....	22
1.7.2. Operadores relacionales .....	22
1.7.3. Operadores lógicos .....	23
1.7.4. Operadores funcionales .....	25
1.7.5. Operaciones con cadenas (strings) .....	25
1.8. PROGRAMA EDITOR .....	26
1.9. TECLAS DE FUNCION .....	30

# SEGUNDA PARTE

## COMANDOS, SENTENCIAS Y FUNCIONES DEL BASIC MSX

<b>1. Ayuda al programador</b> .....	33	• KEY .....	52
• AUTO .....	33	• TIME .....	52
• DELETE .....	34		
• LIST .....	35	<b>4. Entrada/Salida</b> .....	53
• LLIST .....	35	• READ .....	54
• NEW .....	35	• DATA .....	54
• RENUM .....	36	• RESTORE .....	55
• KEY LIST .....	37	• INPUT .....	56
		• LINE INPUT .....	58
<b>2. Programación</b> .....	39	• LET .....	58
• RUN .....	39	• MID\$ = Y\$ .....	59
• STOP .....	40	• PRINT .....	59
• CONT .....	40	• PRINT USING .....	60
• END .....	40	• LPRINT .....	64
• TRON .....	40	• LPRINT USING .....	64
• TROFF .....	40	• LPOS .....	64
• FOR/NEXT .....	41	• SWAP .....	64
• GOTO .....	42	• INKEY\$ .....	65
• IF... THEN... ELSE .....	43	• INPUT\$ .....	65
• IF... GOTO... ELSE .....	43	• FRE .....	65
• GOSUB .....	44	• POKE .....	66
• RETURN .....	44	• PEEK .....	66
• ON... GOTO .....	44	• VPOKE .....	66
• ON... GOSUB .....	45	• VPEEK .....	66
		• INP .....	67
<b>3. Definición e inicialización</b> .....	47	• OUT .....	67
• CLEAR .....	47	• WAIT .....	67
• DIM .....	48	• STICK .....	68
• ERASE .....	48	• STRIG .....	68
• DEFINT .....	49		
• DEFSNG .....	49	<b>5. Interrupciones</b> .....	69
• DEFDBL .....	49	• ON KEY GOSUB .....	69
• DEFSTR .....	50	• KEY ON/OFF/STOP .....	70
• DEFFN .....	50	• ON STRIG GOSUB .....	72
• DEFUSR .....	51	• STRIG ON/OFF/STOP .....	72
• USR .....	51	• ON STOP GOSUB .....	72
		• STOP ON/OFF/STOP .....	73

• ON SPRITE GOSUB .....	73	<b>9. Proceso de Errores</b> .....	89
• SPRITE ON/OFF/STOP ..	73	• ON ERROR GOTO .....	89
• ON INTERVAL = < > GOSUB .....	73	• ERLERR .....	90
• INTERVAL ON/OFF/STOP .....	73	• RESUME .....	90
<b>6. Funciones numéricas</b> .....	75	• ERROR .....	90
• ABS .....	75	<b>10. Pantalla y Gráficos</b> .....	93
• INT .....	76	• SCREEN .....	93
• FIX .....	76	• WIDTH .....	94
• SGN .....	76	• CLS .....	95
• CDBL .....	76	• COLOR .....	95
• CSNG .....	77	• LOCATE .....	95
• CINT .....	77	• TAB .....	96
• EXP .....	77	• SPC .....	96
• LOG .....	77	• POS .....	96
• SQR .....	78	• CSRLIN .....	97
• SIN .....	78	• KEY ON/OFF .....	97
• COS .....	78	• PSET .....	97
• TAN .....	79	• PRESET .....	97
• ATN .....	79	• LINE .....	98
• RND .....	79	• CIRCLE .....	98
<b>7. Conversión de códigos</b> .....	81	• PAINT .....	99
• ASC .....	81	• DRAW .....	100
• CHR\$ .....	81	• SPRITES\$ .....	100
• BIN\$ .....	82	• PUT SPRITE .....	100
• OCT\$ .....	82	• POINT .....	100
• HEX\$ .....	82	• BASE .....	100
• VAL .....	83	<b>11. Sonido</b> .....	101
• STR\$ .....	83	• BEEP .....	101
<b>8. Cadenas de caracteres</b> .....	85	• SOUND .....	101
• RIGHT\$ .....	85	• PLAY .....	103
• LEFT\$ .....	86	• PLAY (N) .....	103
• MID\$ .....	86	<b>12. Almacenamiento en cassette</b> 105	
• STRING\$ (N,X) .....	86	• CSAVE .....	105
• STRING\$ (N,X\$) .....	86	• CLOAD .....	106
• SPACE\$ .....	87	• CLOAD? .....	106
• INSTR .....	87	• SAVE .....	107
• LEN .....	87	• LOAD .....	107
		• BSAVE .....	107
		• BLOAD .....	108
		• MERGE .....	108
		• MOTOR ON/OFF .....	108

<b>13. Ficheros</b> .....	111
• OPEN # .....	111
• MAXFILES .....	112
• PRINT # .....	112
• PRINT # USING .....	112
• INPUT\$(n, #) .....	113
• LINE INPUT # .....	114
• EOF .....	114
• CLOSE # .....	114

## ANEXOS:

<b>1. GRAFICOS</b> .....	119
Introducción .....	119
Macro lenguaje para gráficos .....	119
Gráficos SPRITE .....	125
<b>2. SONIDO</b> .....	135
Introducción .....	135
Macro lenguaje musical .....	135
<b>3. INFORMACION TECNICA</b> .....	139
Mapa de memoria .....	141
Procesador de video .....	153
Conectores .....	160
Cassette .....	160
Audio/Video .....	161
Joystick .....	161
Impresora .....	161
Cartucho .....	162
RGB .....	163
<b>4. PROGRAMAS</b> .....	165
Demostración de los comandos gráficos .....	165
Programa de gráficos .....	166
Simulación rebotes de una pelota .....	166
Cuadros abstractos .....	166
Círculos de colores .....	167
Flores .....	167
Dibujo de círculos de colores aleatorios en forma senoidal .....	168
'Estrellas' .....	168
Programa demostración de gráficos 1 .....	169
Programa demostración de gráficos 2 .....	170

Programa de gráficos:	
«SPRITES»	
Movimiento del cañon → Cursores	
Disparos → Barra de espacio .....	171
Programa de música .....	173
Organo musical .....	173
Himno .....	175
Rock .....	176
Programa para hacer una copia de la pantalla	
(SCREEN 0) por impresora .....	177
Programa para hacer una copia de la pantalla	
(SCREEN 1) por impresora .....	177
Programa para hacer una copia de la pantalla	
(SCREEN 2/3) con el Plotter PRN-C41 de SONY .....	177
Programa para utilizar con el Plotter PRN-C41 de SONY:	
—Círculo .....	178
—Cuadrícula .....	179
Juego: Laberinto .....	179
Programa para dibujar con los cursores .....	182
<b>5. MENSAJES Y CODIGOS DE ERROR .....</b>	<b>185</b>



# **PRIMERA PARTE**

## **INFORMACION GENERAL DEL BASIC MSX**

PRIMERA PARTE  
INFORMACION GENERAL  
DEL BASIC MSX

## 1.0. INTRODUCCION

El **BASIC MSX** es una versión ampliada del Microsoft Standard Basic versión 4.5, e incluye soportes para gráficos, música y diversos periféricos

Fue diseñado para seguir el **GW-BASIC**, estandard en el mundo de los microprocesadores de 16 bits. Pero el mayor esfuerzo fue hacer todo el sistema lo más flexible y ampliable posible.

El **BASIC MSX** trabaja con 14 dígitos para ofrecer doble precisión en las funciones matemáticas, por lo que no se generarán más esos extraños errores de redondeo que confunden a los usuarios.

## 1.1. MODOS DE FUNCIONAMIENTO

Una vez inicializado el **BASIC MSX**, aparece rápidamente el OK. «OK», indica que el sistema está dispuesto para aceptar cualquier comando. Es ahora cuando el **BASIC MSX** puede ser usado en modo directo o indirecto.

En el modo directo los comandos no están precedidos por número de línea y se ejecutan al ser introducidos. Los resultados de operaciones aritméticas y lógicas aparecen inmediatamente en la pantalla y son almacenados para su uso posterior, pero las instrucciones desaparecen tras su ejecución (no quedan almacenadas en memoria).

El modo directo es útil para utilizar el **BASIC MSX** como calculadora, para rápidas operaciones que no requieren un programa completo.

El modo indirecto se utiliza para introducir programas. Las líneas del programa están precedidas por números y son almacenadas en memoria. El programa almacenado en memoria será ejecutado al introducir el comando **RUN**.

## 1.2. FORMATO DE LAS LINEAS

En una línea de programa, pueden situarse varias instrucciones, siempre que estén separadas por dos puntos:

## 10 instrucción **BASIC**: instrucción **BASIC**

Toda línea de programa comienza siempre con un número de línea y termina al pulsar **RETURN**, pudiendo contener un máximo de 255 caracteres.

### 1.2.1. Número de línea

Cada línea de un programa **BASIC MSX**, comienza con un número.

Los números de línea indican el orden en que han sido almacenadas en memoria las líneas del programa.

También son utilizados como referencia en la edición del programa y en las sentencias de bifurcación.

Los números de línea deben estar comprendidos entre 0 y 65.529.

## 1.3. CONJUNTO DE CARACTERES

El conjunto de caracteres del **BASIC MSX** está formado por: caracteres alfabéticos, caracteres numéricos, caracteres especiales y caracteres gráficos.

Los caracteres alfabéticos son las letras del alfabeto.

Los caracteres numéricos son los dígitos del 0 al 9.

Los siguientes caracteres especiales están reconocidos por el **BASIC MSX**:

Caracter	Acción
=	signo de equivalencia
+	suma
-	resta
*	multiplicación

Carácter	Acción
/	división
^	símbolo exponencial o acento
`	circunflejo
(	paréntesis izquierdo
)	paréntesis derecho
%	porcentaje
\$	signo de dólar
!	exclamación
{	corchete izquierdo
}	corchete derecho
.	punto o punto decimal
'	apóstrofe
:	punto y coma
::	dos puntos
&	and
?	signo de interrogación
<	menor que
>	mayor que
¥	signo del yen

## 1.4. CONSTANTES

Las constantes son los valores que el **BASIC MSX** usa durante la ejecución de un programa. Hay dos tipos de constantes: numéricas y alfanuméricas (string).

Una constante alfanumérica es una sucesión de hasta 255 caracteres alfanuméricos insertados entre comillas.

"HOLA" "25.000.000\$" "Número de empleados".

Las constantes numéricas son números positivos o negativos, y no pueden contener comas (El punto realiza la función de la coma). Hay seis tipos de constantes numéricas:

1. **Constantes enteras:** todo número entre -32768 y 32767. Las constantes enteras no contienen decimales
2. **Constantes con punto decimal fijo:** números reales positivos o negativos que contengan decimales.

3. **Constantes con punto decimal**, número positivos o negativos en forma exponencial. El margen para estas constantes está entre  $10^{-64}$  y  $10^{+63}$ .

235.988 E-7 = 0.0000235988  
2359 E6 = 2359000000

4. **Constantes hexadecimales**: números hexadecimales, con el prefijo & H

& H76  
& H32 F

5. **Constantes octales**: números octales, con el prefijo & O ó &

& O 347  
& 347

6. **Constantes binarias**: números binarios, con el prefijo & B

& B 01110110  
& B 11100111

#### 1.4.1. Constantes numéricas de simple y doble precisión

Las constantes numéricas de simple precisión están almacenadas con 6 dígitos y son impresas con hasta 6 dígitos de precisión. Las constantes numéricas de doble precisión están almacenadas con 14 dígitos y son impresas con hasta 14 dígitos de precisión.

Una constante de simple precisión es toda constante numérica que cumple una de las siguientes características:

1. Forma exponencial usando E
2. Signo de admiración (!)

-1.09 E-06

22.5!

Una constante de doble precisión es toda constante numérica que cumple una de las siguientes características:

1. Todo número no exponencial (usando E) sin signos de admiración.
2. Número exponencial usando la letra D
3. Número con el signo #

3489

345962811

-1.09432 D-06

3489.0 #

## 1.5. VARIABLES

Las variables son nombres usados para representar valores. El valor de una variable puede ser asignado explícitamente por el programador o puede ser el resultado de unos cálculos en el programa. Al inicializar el sistema, todas las variables tendrán asignado el valor cero (variables numéricas) ó «null» (variables alfanuméricas).

### 1.5.1. Nombres de variables y Declaración de caracteres

Los nombres de variables en el **BASIC MSX** pueden ser de cualquier longitud, pero solo son significativos 2 caracteres. Estas variables pueden contener letras y números pero el primer carácter debe ser siempre una letra.

Una variable no puede ser una palabra reservada y no puede contener palabras reservadas. Las palabras reservadas incluyen todos los comandos del **BASIC MSX**, nombres de funciones, ordenes y nombres operativos. Por ejemplo, si una variable comienza por **FN**, se supondrá que es la llamada de una función definida por el usuario y no el nombre de una variable.

Las variables pueden representar un valor numérico o una cadena de caracteres. Las variables alfanuméricas se representan con el símbolo \$ como último carácter:

```
A$ = "VENTAS"
```

Las variables numéricas pueden ser números enteros, de simple o doble precisión. Los símbolos que identifican esas variables son:

% variable entera  
! variable de simple precisión.  
# variable de doble precisión.

Pi #	doble precisión
Minimum!	simple precisión
Limit %	variable entera
N\$	variable alfanumérica
ABC	doble precisión

Existe un segundo método por el cual pueden definirse los tipos de variables utilizando las sentencias **DEFINT**, **DEFSTR**, **DEFSGN** y **DEFDBL** en el transcurso de un programa.

### 1.5.2. Variables matriciales (variables subindicadas).

El **BASIC MSX**, permite trabajar con variables matriciales para designar elementos de una matriz.

Las matrices deben declararse por medio de una sentencia de

dimensionado (DIM) en el programa, pero ello no es imprescindible si los subíndices no superan el rango de 0 a 10.

El nombre de una variable con subíndice es siempre variable y va seguido por una o más expresiones entre paréntesis. La expresión entre paréntesis indica la posición de los datos en la Matriz.

A \$(4): variable subíndicada de una dimensión.

A (2,3): variable subíndicada de dos dimensiones.

A (2,3,.....,6): variable subíndicada de varias dimensiones.

Una matriz puede tener hasta 255 dimensiones. El número máximo de elementos está determinado por la capacidad de memoria.

### 1.5.3. Requisitos de espacio

La siguiente tabla, indica el número de bytes ocupados por los valores representados en los distintos tipos de variables:

<b>Variables numéricas:</b>	<b>Tipo</b>	<b>Bytes</b>
	entera	2
	simple precisión	4
	doble precisión	8
<b>Matrices:</b>	entera	2 por elemento
	simple precisión	4 por elemento
	doble precisión	8 por elemento
<b>Variables alfanuméricas:</b>	3 bytes más el contenido de la cadena	

## 1.6. CONVERSION DE TIPOS

Cuando es necesario el BASIC MSX convierte una constante numérica de un tipo en otra. He aquí unos ejemplos:

1. Si una constante numérica de un tipo se iguala a una variable numérica distinta, el número será almacenado como se indica en la variable.

```
10 A % = 23,42
20 PRINT A %
RUN
23
```

Si una variable string se iguala un valor numérico o viceversa, aparecerá el siguiente mensaje de error: "Type mismatch".

2. Durante el cálculo de una operación todos los operandos son convertidos a un único grado de precisión y lo mismo sucede con el resultado.

```
10 D = 6/7!
20 PRINT D
RUN
.85714285714286
```

La operación y resultado son en doble precisión.

```
10 D! = 6/7
20 PRINT D!
RUN
.857143
```

La operación se ha hecho en doble precisión pero el resultado aparece en D!, con simple precisión y redondeado.

3. Los operandos pueden ser convertidos en enteros y obtener un resultado en números enteros. Los operandos deben estar comprendidos entre  $-32768$  y  $32767$  para que no exista un error de «overflow».
4. Cuando un valor con punto decimal flotante se convierte en un entero, la parte decimal desaparece.

```
10 C% = 55.88
20 PRINT C %
RUN
55
```

5. Si una variable de doble precisión se asigna a un valor de simple precisión solo serán válidos los primeros 6 dígitos.

```
10 A! = SQR (2)
20 B = A!
30 PRINT A!, B
RUN
1.41421    1.41421
```

## 1.7. EXPRESIONES Y OPERADORES

Una expresión puede ser una cadena o una constante numérica, una variable, una combinación de constantes y variables con operandos que producen un valor.

Los operadores del **BASIC MSX** pueden ser divididos en 4 clases: aritméticos, relacionales, lógicos y funcionales.

### 1.7.1. Operadores aritméticos

Operador	Operación	Expresión
^	Exponencial	$X \sim Y$
-	Negación	$-X$
*, /	Multiplicación, División	$X*Y$ $X/Y$
+, -	Adición, sustracción	$X+Y$ $X-Y$

Para cambiar el orden de realización de las operaciones, se usa el paréntesis.

Las operaciones en el interior del paréntesis se realizan en primer lugar. Dentro del paréntesis, se mantiene el orden usual de las operaciones: multiplicación y división en primer lugar y tras estas, suma y resta.

#### 1.7.1.1. División entera y Resto

Dos operandos adicionales están disponibles en el **BASIC MSX**

La división entera se indica con el símbolo de Yen. Los operandos se convierten a enteros (entre -32768 y 32767) antes de efectuar la división y el cociente es redondeado a entero.

$$10/4 = 2$$
$$25.68/6.99 = 4$$

MOD, da el valor entero del resto de la división.

$$10 \text{ MOD } 4 = 2 \text{ (} 10/4 = 2 \text{ con el resto } 2\text{)}$$
$$25.63 \text{ MOD } 6.99 = 4 \text{ (} 25/6 = 4 \text{ con el resto } 1\text{)}.$$

#### 1.7.1.2. Overflow y división por cero

Si durante el cálculo de una operación, se encuentra la división por cero aparece el mensaje de error «División by zero» y termina la ejecución del programa.

Asimismo, cuando se produce una sobrecarga de datos aparece el mensaje de error «Overflow».

#### 1.7.2. Operadores relacionales

Son usados para comparar dos valores. El resultado de la comparación es verdadero (-1) o falso (0)

Operador	Operación	Expresión
=	Igualdad	$X = Y$
< >	Desigualdad	$X <> Y$
<	Menor que	$X < Y$
>	Mayor que	$X > Y$
< =	Menor o igual que	$X > = Y$
> =	Mayor o igual que	$X > = Y$

El signo igual se usa también para asignar un valor a una variable.

Cuando se combinan operadores aritméticos y relacionales en una expresión, el aritmético se ejecutará siempre en primer lugar.

$$X + Y < (T-1)/2$$

En primer lugar se realizarán las operaciones:  $X + Y$  y  $(T-1)/2$ , luego se compararán los resultados obtenidos.

### 1.7.3. Operadores lógicos

Realizan operaciones lógicas, bit a bit. El resultado puede ser cierto (1) o falso (0). En una expresión, las operaciones lógicas son ejecutadas tras las aritméticas y las relacionales. La función que realizan los operadores lógicos NOT, AND, OR, XOR, EQU e IMP se resume en la siguiente tabla:

**Operadores lógicos del BASIC MSX.**

NOT	$\frac{X}{1}$ $\frac{X}{0}$	$\frac{\text{NOT } X}{0}$ $\frac{\text{NOT } X}{1}$	
AND	$\frac{X}{1}$ $\frac{X}{1}$ $\frac{X}{0}$ $\frac{X}{0}$	$\frac{Y}{1}$ $\frac{Y}{0}$ $\frac{Y}{1}$ $\frac{Y}{0}$	$\frac{X \text{ AND } Y}{1}$ $\frac{X \text{ AND } Y}{0}$ $\frac{X \text{ AND } Y}{0}$ $\frac{X \text{ AND } Y}{0}$
OR	$\frac{X}{1}$ $\frac{X}{1}$ $\frac{X}{0}$ $\frac{X}{0}$	$\frac{Y}{1}$ $\frac{Y}{0}$ $\frac{Y}{1}$ $\frac{Y}{0}$	$\frac{X \text{ OR } Y}{1}$ $\frac{X \text{ OR } Y}{1}$ $\frac{X \text{ OR } Y}{1}$ $\frac{X \text{ OR } Y}{0}$
XOR	$\frac{X}{1}$ $\frac{X}{1}$ $\frac{X}{0}$ $\frac{X}{0}$	$\frac{Y}{1}$ $\frac{Y}{0}$ $\frac{Y}{1}$ $\frac{Y}{0}$	$\frac{X \text{ XOR } Y}{0}$ $\frac{X \text{ XOR } Y}{1}$ $\frac{X \text{ XOR } Y}{1}$ $\frac{X \text{ XOR } Y}{0}$



```

10 OR 10 = 10   10 = binario 1010
                 así 1010 OR 1010 = 1010

-1 OR -2 = 1    -1 = binario 1111111111111111
                 -2 = binario 1111111111111111
                 así -1 OR -2 = -1 (binario 1111111111111111)
NOT X = -(X+1)

```

#### 1.7.4. Operadores funcionales

El **BASIC MSX**, tiene funciones intrínsecas residente en el sistema, como **SQR** (raíz cuadrada) ó **SIN** (seno). También pueden definirse funciones escritas por el programador con el comando, **DEF FN**.

#### 1.7.5. Operaciones con cadenas (strings)

Los strings pueden ser encadenados usando el signo +

```

10 A$ = "FILE": B$ = "NAME"
20 PRINT A$ + B$
30 PRINT "NEW" + A$ + B$
RUN
FILE NAME
NEW FILENAME

```

Los strings pueden ser comparados usando los mismos signos de comparación que los usados con números

```
= < < > <= >=
```

Las comparaciones entre strings, se efectúan carácter a carácter utilizando el código **ASCII**. Si todos los códigos **ASCII**

son iguales, los string son iguales. Si durante la comparación de strings se llega al final de uno de ellos, el string más corto será el más pequeño. Los espacios en blanco son significativos.

```
"A" < "AB"  
"FILENAME" = "FILENAME"  
"X&" > "X ="  
"CL  " > "CL"  
"kg" < "KG"  
"SMYTH" < "SMYTHE"  
B$ < "9/12/83" donde B$ = "8/12/83"
```

Las comparaciones de strings pueden ser usadas para comprobar el valor de un string ó para alfabeticar strings. Todas las constantes utilizadas en las expresiones de comparación deben ir entre comillas.

## 1.8. PROGRAMA EDITOR

El programa editor, permite al usuario realizar todos los cambios necesarios dentro del programa (borrar o añadir líneas de programa, insertar instrucciones,...) utilizando para ello los cursores y las teclas **INSERT** (insertar) y **DELETE** (borrado). El usuario puede situarse rápidamente en cualquier punto de la pantalla y hacer las correcciones necesarias.

### Programas

El Editor de pantalla está en funcionamiento continuamente desde la aparición de «OK» hasta que se ejecuta un programa mediante **RUN**. El Editor procesa todas las líneas de texto introducidas, considerando una línea de programa toda aquella que comience por un número.

Si se intenta borrar una línea inexistente aparecerá el mensaje de error: «Undefined line number».

El Editor procesará los comandos del programa en los siguientes casos:

1. **Adición de una nueva línea al programa:** El número de líneas debe estar comprendido entre 0 y 65529 y debe ir seguido de un carácter como mínimo.
2. **Modificación de una línea ya existente:** la línea existente queda reemplazada por el nuevo texto introducido.
3. **Borrado de una línea:** Se produce si la nueva línea contiene sólo el número de línea que se desea borrar.
4. **Se produce un error.**

Si se pretende introducir una nueva línea de programa y no hay suficiente espacio en memoria, se imprimirá el mensaje de error «Out of memory».

Una línea de programa puede contener varias instrucciones, siempre que estén separadas por dos puntos (:). El número máximo de caracteres por línea de programa es 250.

### **Edición de Programas**

Mediante el comando **LIST** nos aparecerán todas las instrucciones del programa, pudiendo así ser editadas. El texto puede ser modificado moviendo el cursor hasta el lugar donde se precisa hacer el cambio. Esta modificación puede realizarse efectuando una de las siguientes acciones:

1. Escribiendo directamente sobre el carácter equivocado.
2. Borrando los caracteres a la derecha del cursor.
3. Borrando los caracteres a la izquierda del cursor.
4. Insertando caracteres.
5. Añadiendo caracteres al final de la línea.

Los cambios en una línea quedan grabados cuando se pulsa la tecla **RETURN**.

### **Funciones del editor de pantalla**

La siguiente tabla muestra los códigos hexadecimales del **BASIC MSX**

Tabla 1. Funciones del control del **BASIC MSX**. Pulsando a un tiempo la tecla **CTRL** y otra determinada, se ejecuta una operación especial.

Código hexadecimal	CTRL +	Operación realizada
01	A	—
02	B	Mueve el cursor al principio de la palabra precedente.
03	C	Abandona la espera de un input y la numeración automática de líneas.
04	D	—
05	E	Borra texto entre el cursor y el final de la línea.
06	F	Mueve el cursor al principio de la palabra siguiente.
07	G	Sonido Bip.
08	H	Misma función que tecla BS.
09	I	Misma función que tecla TAB.
0A	J	Mueve el cursor a la línea posterior.
0B	K	Misma función que tecla HOME.
0C	L	Limpia pantalla.
0D	M	Misma función que tecla RETURN.
0E	N	Mueve el cursor a la posición posterior a la del último carácter de la línea.
0F	O	—
10	P	—
11	Q	—
12	R	Misma función que tecla INS
13	S	—
14	T	—
15	U	Borra los caracteres de la línea hasta la posición del cursor.
16	V	—
17	W	—
18	X	Misma función que tecla SELECT.
19	Y	—
1A	Z	—
1B	[	Misma función que tecla ESC.
1C	/	Mueve el cursor a la derecha.
1D	]	Mueve el cursor a la izquierda.
1E	^	Mueve el cursor arriba.
1F	—	Mueve el cursor abajo.

### Funciones de teclas especiales

Tecla TAB: Mueve el cursor 8 posiciones a la derecha. Todos los caracteres existentes en esas posiciones son borrados.

Tecla RETURN: Debe pulsarse siempre que se haya terminado de entrar una línea.

Tecla ESC: La función de esta tecla viene determinada por el software utilizado. Para el **BASIC MSX**, no tiene utilidad concreta.

Tecla STOP: Al presionar la tecla se interrumpe la ejecución de un programa o de un listado. Al presionarla de nuevo, se reanuda la ejecución. Pulsando al mismo tiempo CTRL, y STOP se termina la ejecución del programa con el siguiente mensaje Break in (nº instrucción).

Tecla SELECT: La función de esta tecla viene determinada por el software utilizado. Para el **BASIC MSX**, no tiene utilidad concreta.

Tecla INSERT: Al pulsarla, se reduce el tamaño del cursor y pueden introducirse los caracteres deseados en la posición del cursor, mientras los caracteres a la derecha del mismo van desplazándose a medida que se introducen los nuevos. Pulsando la tecla INSERT de nuevo, el cursor vuelve a su posición normal y termina la función de inserción. Pulsando las teclas de movimiento del cursor o pulsando RETURN, también termina la función de inserción.

Tecla HOME: Mueve el cursor al extremo superior izquierdo de la pantalla.

Tecla DEL: Borra el carácter situado en la posición del cursor. Los caracteres siguientes se desplazan una posición a la izquierda.

Tecla BS: Al pulsarla, el cursor se desplaza a su izquierda, borrando el carácter que esté en esa posición.

Tecla SHIFT: Pulsándola al mismo tiempo que cualquier tecla, aparece en pantalla el símbolo superior izquierdo de la tecla en cuestión.

Tecla CAP: Las letras que aparecen en pantalla son mayúsculas, pero los números y símbolos no varían.

Tecla CODE: Pulsando CODE y cualquier tecla, aparece el símbolo inferior izquierdo de la tecla. Si pulsamos, además, SHIFT, aparece el símbolo superior izquierdo.

Tecla GRAPH: Pulsándola al mismo tiempo que cualquier tecla, aparece el símbolo inferior derecho de la tecla. Si pulsamos además, SHIFT, aparece el símbolo superior derecho.

## 1.9. TECLAS DE FUNCION

El **BASIC MSX**, dispone de 10 funciones predefinidas en las teclas de función F1/F10. El contenido de estas teclas se visualiza en el margen inferior de la pantalla pudiendo ser reprogramadas mediante el comando **KEY**.

Las funciones predefinidas son:

F1	color
F2	auto
F3	goto
F4	list
F5	run

Al pulsar la tecla **SHIFT**, tendremos:

F6	color 15,4,7. RETURN
F7	cloud "
F8	cont. RETURN.
F9	list. RETURN.
F10	cls: run. RETURN.

# **SEGUNDA PARTE**

**COMANDOS, SENTENCIAS  
Y FUNCIONES DEL BASIC MSX**

SEGUNDA PARTE

COMANDOS, SENTENCIAS  
Y FUNCIONES DEL BASIC MAX

## 1. COMANDOS DE AYUDA AL PROGRAMADOR

- **AUTO**
- **DELETE**
- **LIST**
- **LLIST**
- **NEW**
- **RENUM**
- **KEY LIST**

**AUTO** < num. línea>, <incremento>

Numera automáticamente las líneas de un programa.

- < núm. línea>: línea de inicio del programa.
- < incremento>: salto entre líneas.

**AUTO** empieza la numeración en <núm. línea> y va numerando las siguientes según el valor de <incremento>.

Si **AUTO** genera un número de línea utilizado anteriormente con otra instrucción, aparecerá un asterisco. (\*) Pulsando **RETURN**, conservara el contenido de la línea y generara una nueva. De lo contrario cualquier entrada que se efectúe en esa línea, anulara la anterior.

Para anular el comando **AUTO**, pulsar **CTRL - STOP** ó **CTRL-C**.

AUTO: Numera desde la línea 10 con incrementos de 10:

10  
20  
30  
:

AUTO 120: Numera desde la línea 120 con incrementos de 10:

120  
130  
140  
:

AUTO 100,5: Numera desde la línea 100 con incrementos de 5:

100  
105  
110  
:

AUTO,5: Numera desde la línea 0, con incrementos de 5:

0  
5  
10  
:

**DELETE** < núm. línea a> — <núm. línea b>

Borra las líneas de programa comprendidas entre <núm. línea a> y <núm. línea b>.

- <núm. línea a>: línea de inicio de borrado
- <núm. línea b>: última línea a borrar.

Si el número de línea especificado en <núm. línea b> no existe en el programa, se producirá el error: «Illegal function call».

**DELETE 10-60:** Borra desde la línea 10 hasta la 60. (ambas inclusive).

**DELETE 20:** Borra la línea 20.

**NOTA.-** También puede borrar una sola línea, escribiendo el núm. de línea y pulsando **RETURN**:

20 (RETURN).

**LIST** <nº línea a> - <nº línea b>  
Lista el programa por pantalla.

- <nº línea a>: línea de inicio del listado.
- <nº línea b>: línea final del listado.

Podemos detener momentáneamente el listado, pulsando **STOP**. Pulsando **STOP** de nuevo, continuará el listado.

Para anular la función **LIST**, pulsar **CTRL-STOP**.

<b>LIST</b>	: lista todo el programa.
<b>LIST 10</b>	: lista la línea 10 del programa,
<b>LIST 10-50</b>	: lista desde la línea 10 hasta la 50. (ambas inclusive).
<b>LIST -50</b>	: lista hasta la línea 50.
<b>LIST 50-</b>	: lista desde la línea 50 hasta el final del programa.

**LLIST** <nº línea a> - <nº línea b>  
Lista el programa por impresora. Funciona igual que **LIST**. (ver **LIST**).

**NEW** Borra el contenido de la memoria **RAM** de usuario.

**RENUM** <n° línea nuevo>, <n° línea actual>, <incremento>  
Cambia la numeración de las líneas de un programa.

- <n° línea nuevo>: línea inicial de la nueva numeración.
- <n° línea actual>: línea inicial de la numeración actual.
- <incremento>: Salto entre líneas que se utilizará en la nueva numeración. Si no se especifica, será 10.

**RENUM** cambia también las referencias de los números de línea que acompañan a **GOTO**, **GOSUB**, **THEN**, **ELSE**, **ON GOTO**, **ON GOSUB** y **ERL**.

**RENUM** : Renumerar el programa desde la línea 10 con incrementos de 10.

**RENUM 1000** : Renumerar y trasladar el programa a partir de la línea 1000 con incrementos de 10.

**RENUM 100, 60, 5** : Traslada a la línea 100 numerando con incrementos de 5. el programa que estaba en la línea 60:

```
10 REM $ EJEMPLO RENUM $
20 PRINT "HIT BIT"
30 PRINT "BASIC"
50 PRINT "MSX"
60 A = 8
70 PRINT A
77 B = 10
85 PRINT B
90 GOTO 70
```

```
RENUM 1000, 60, 5
```

```
10 REM $ EJEMPLO RENUM $
```

```
20 PRINT "HIT BIT "
```

```
30 PRINT "BASIC "
```

```
50 PRINT "MSX "
```

```
1000 A = 8
```

```
1005 PRINT A
```

```
1010 B = 10
```

```
1015 PRINT B
```

```
1020 GOTO 1005
```

### **REM** <comentarios>

Es un comando no ejecutable por el ordenador que permite hacer comentarios dentro de un programa.

●<comentarios>: Comentarios introducidos para diferenciar las distintas partes de un programa, siendo útil únicamente en los listados ya que **REM** no es ejecutado por el ordenador.

```
10 REM PROGRAMA PRINCIPAL
```

```
100 REM SUBROUTINA 1
```

```
600 REM CONTADOR
```

### **KEY LIST**

Lista el contenido de todas las teclas de función (F1-F10).

#### KEY LIST (RETURN)

color

auto

goto

list

```
run
color 15,4,7
load''
cont
list.
run
```

Color corresponde a la tecla F1, Auto a F2. Goto a F3 y así sucesivamente. hasta F10.

## 2. COMANDOS DE PROGRAMACION

- RUN
- STOP
- CONT
- END
- TRON
- TROFF
- FOR/NEXT
- GOTO
- IF.. THEN.. ELSE
- IF.. GOTO.. ELSE
- GOSUB
- RETURN
- ON... GOTO
- ON... GOSUB

**RUN** <n° línea>

Ejecuta el programa a partir del número de líneas especificado.

- <n° línea> : número de línea donde empezará la ejecución del programa. Si no se especifica, ejecutará el programa desde el principio.

**RUN** : Ejecuta todo el programa.

**RUN 120** : Ejecuta el programa a partir de la línea 120.

## **STOP**

Detiene la ejecución de un programa y se pone a la espera de nuevas órdenes.

Cuando el ordenador encuentra un **STOP**, imprimirá el siguiente mensaje:

**BREAK IN** × (× es el núm. de línea donde se encuentra el **STOP**).

A diferencia del comando **END**, **STOP** no cierra ficheros.

La ejecución del programa puede continuar utilizando el comando **CONT**. (ver **CONT**.).

## **CONT**

Continúa la ejecución de un programa después de un **BREAK** o **STOP**. No podrá utilizarse en los siguientes casos:

1. Cuando la interrupción sea debida a un error. (ver código de Errores).
2. Cuando se efectúe cualquier modificación dentro del programa.
3. Cuando se incorpore una nueva línea en el programa.

Al utilizar **CONT** en cualquiera de estos casos se imprimirá el mensaje de error: «Can't continue».

## **END**

Termina la ejecución de un programa, cierra todos los ficheros y se pone a la espera de nuevas órdenes. Es opcional.

## **TRON**

Visualiza en pantalla los números de línea por los que pasa el programa durante su ejecución, facilitando así el descubrimiento y la corrección de errores.

Puede introducirse como instrucción directa antes de la ejecución del programa.

## **TROFF**

Anula la función de **TRON**.

**FOR** <variable> = <valor inicial> **TO** <valor final> **STEP** <incremento>

Se utiliza conjuntamente con la sentencia **NEXT** para establecer una sección de programa que se repita un número dado de veces. (Bucle **FOR NEXT**).

- <variable>: Debe ser numérica. Inicialmente tendrá asignado el <valor inicial>, incrementándose, según el valor especificado en <incremento> hasta alcanzar el <valor final>.
- <valor inicial>: Valor inicial que tendrá la <variable>. Puede ser un número ó una expresión matemática en la que aparezcan valores numéricos y/o variables relacionados por operadores.
- <valor final>: Valor final que tomara la <variable>.
- <incrementos>: Define el valor del paso o salto que, sucesivamente, incrementará o decrementará el valor de la <variable> según sea positivo o negativo. Si no se especifica, el incremento será 1. (**STEP1**).

**NEXT** <variable>.

Es la última instrucción de un bucle **FOR NEXT**.

- <variable> Nombre de variable que identifica el bucle **FOR NEXT**. Debe corresponder con la variable especificada en **FOR**.

**NEXT**, puede adoptar 3 formas distintas:

- NEXT** : cierra el último bucle en caso de que hubiese varios.
- NEXT A** : cierra el bucle identificado con la variable **A**.
- NEXT A,B** : cierra primero el bucle identificador con la variable **A** y luego el identificado con la variable **B**.

1. Imprime el cuadrado de los 100 primeros números:

```
10 FOR I = 1 TO 100
20 PRINT I^2
30 NEXT I
```

2. Imprime todos los números pares de 0 a 100 en modo decreciente.

```
10 FOR I = 100 TO 2 STEP - 2
20 PRINT I
30 NEXT I
```

3. Lee una lista de 10 números con **READ/DATA** y los imprime en pantalla.

```
10 FOR X = 1 TO 10
20 READ A (X)
30 PRINT A (X)
40 NEXT X
60 DATA 100, 45, 320, 65, 43, 22, 4, 8, 9, 2.
```

Dentro de un mismo programa pueden utilizarse bucles **FOR/NEXT** anidados, ejecutándose unos dentro de otros. (no pueden entrelazarse).

#### Bucles anidados correctos

```
10 FOR A = 1 TO 2
  20 FOR B = 1 TO 5
    30 PRINT A,B
  40 NEXT B
50 NEXT A
60 END
```

#### Bucles anidados incorrectos

```
10 FOR A = 1 TO 2
  20 FOR B = 1 TO 5
    30 PRINT A,B
  40 NEXT A
  50 NEXT B
60 END
```

**GOTO** <num. línea>

Salto incondicional al número de línea especificado.

- <núm. línea>: número de línea en el que continuara la ejecución del programa.

Es caso de especificar un número de línea que no existe en el programa se imprimirá el siguiente mensaje de error: "Undefined line number".

```
10 GOTO 40
20 END
40 PRINT "VENGO DE LA LINEA 10"
50 GOTO 20
```

**IF** <1ª expresión> **THEN** <2ª expresión> **ELSE** <3ª expresión>

**IF** es un «si» condicional. Es decir, si la <1ª expresión> se cumple, entonces (**THEN**) ejecuta la <2ª expresión>, que puede ser un número de línea, con lo cual pasará el control del programa a dicho número de línea, o bien una instrucción concreta.

Si la <1ª expresión> no se cumple, pasará a ejecutar la <3ª expresión>, que igualmente puede ser un número de línea o una instrucción concreta.

```
10 IF A = B THEN 80 ELSE 140
```

Si A es igual a B, entonces (**THEN**) ve a la línea 80.

De lo contrario (**ELSE**), ve a la 140.

```
10 IF A = B THEN PRINT "A=B" ELSE PRINT "A<>B"
```

Si A es igual a B, entonces (**THEN**) imprime "A = B".

De lo contrario (**ELSE**), imprime "A<>B".

Si no se utiliza **ELSE** y la <1ª expresión> no se cumple, continuará la ejecución en la siguiente línea del programa.

```
10 IF A = B THEN PRINT "A = B"
```

```
20 PRINT "A<>B"
```

Si A es igual a B imprimira "A = B".

De lo contrario, imprimira "A<>B".

**IF** <1ª expresión> **GOTO** <núm. línea> **ELSE** <2ª expresión>

Es similar a **IF THEN ELSE** con la diferencia que **GOTO** irá siempre acompañado de un número de línea.  
(Ver **IF THEN ELSE**)

```
10 IF A = 20 GOTO 60 ELSE C = 0
```

### **GOSUB** <num. línea>

Desvía la ejecución del programa hacia una subrutina.

- <núm. línea>: línea de inicio de la subrutina.

Una subrutina, es un pequeño programa que realiza una función concreta. Si durante un programa se necesita realizar una determinada función varias veces, en lugar de escribir las líneas de programa que realizan dicha función tantas veces como se precise ejecutarla, se escribe una sola vez convirtiéndola en una subrutina.

A lo largo del programa, el ordenador se dirigirá a dicha subrutina tantas veces como se le indique retornando a la ejecución normal del programa principal una vez haya realizado la subrutina en cuestión.

### **RETURN**

Retorno, desde una subrutina, al programa principal. Es la última instrucción de una subrutina, continuando la ejecución del programa en la instrucción siguiente a la que fue llamada mediante **GOSUB**.

```
10 PRINT "EJEMPLO SUBRUTINA"  
20 GOSUB 500  
30 PRINT "VUELVO DE LA SUBRUTINA"  
35 END  
500 PRINT "ESTOY EN LA SUBRUTINA"  
510 RETURN
```

### **ON** <expresión> **GOTO** <lista nº línea>

Salta a un número de línea del programa, determinado por el valor de <expresión>

- <expresión>: variable ó expresión de cuyo valor depende el número de línea donde se efectuara el **GOTO**.
- <lista nº línea>: número de líneas correspondientes a los saltos que efectuará **GOTO**, según el valor de <expresión>. Por

ejemplo, si el valor de <expresión> es 3, se hará un **GOTO** con el 3<sup>er</sup> número de la lista.  
Los números de línea deberán estar separados por coma.

**ON A GOTO 70, 100, 190, 600.**

- Si  $A = 0$ , salta a la instrucción siguiente a **ON..GOTO**
- Si  $A = 1$ , salta a la línea 70.
- Si  $A = 2$ , salta a la línea 100.
- Si  $A = 3$ , salta a la línea 190
- Si  $A = 4$ , salta a la línea 600
- Si  $A > 5$ , salta a la siguiente instrucción a **ON GOTO**.

**ON <expresión> GOSUB <lista n° línea>**

Es similar a **ON GOTO**, con la diferencia de que los saltos se efectuaran a las subrutinas, cuyas líneas de inicio son las indicadas en <lista n° línea>.

**ON A\*5 GOSUB 100, 300**

- Si  $A*5 = 0$ , Salta a la instrucción siguiente a **ON..GOSUB**.
- Si  $A*5 = 1$ , Salta a la subrutina que empieza en la línea 100.
- Si  $A*5 = 2$ , Salta a la subrutina que empieza en la línea 300.
- Si  $A*5 = 3$ , Salta a la instrucción siguiente a **ON..GOSUB**.



### 3. DEFINICION E INICIALIZACION

- CLEAR
- DIM
- ERASE
- DEFINT
- DEFSNG
- DEFDBL
- DEFSTR
- DEF USR
- USR
- KEY
- TIME

#### **CLEAR**

Asigna a todas las variables numéricas el valor cero y borra todos los datos acumulados en las variables alfanuméricas.

Usado como comando, seguido de un número, además de realizar las funciones descritas, en el párrafo anterior, reserva el número especificado en **BYTES** para las cadenas relativas a las variables alfanuméricas.

- |                  |  |
|------------------|--|
| <b>CLEAR</b>     | : Asigna a todas las variables el valor cero.  |
| <b>CLEAR 500</b> | :Asigna a todas las variables el valor cero, y reserva 500 Bytes para las cadenas relativas a las variables alfanuméricas. |

**NOTA:** Inicialmente, el ordenador reserva un espacio de 200 Bytes para cadenas alfanuméricas, pudiendo ser ampliado mediante CLEAR si las necesidades del programa lo requieren.

**DIM** variable, <(dimensión)>

Dimensiona una variable subindicada (numérica o alfanumérica), determinando el número máximo de elementos que podrá contener.

- <variable>: variable a dimensionar.
- <dimensión>: dimensión de la variable (ver apartado 1.5.2. variables subindicadas)

Las variables subindicadas podrán tener uno o más subíndices y por consiguiente, capaces de albergar simultáneamente varios valores.

```
DIM A (10)
DIM A (2,5)
DIM A (2,3,2)
DIM A (2,2,3,4,...)
```

**ERASE** <lista de variables>

Borra las variables dimensionadas especificadas en <lista de variables>, con el fin de poder dimensionarlas de nuevo sin que se produzca el error: "Redimensioned Array".

- <lista de variables>: variables dimensionadas a borrar.

```
10 DIM A (15)
20 FOR I = 0 TO 15
30 A (I) = I + 1
40 NEXT I
50 ERASE A
60 DIM A (20)
```

Sin la instrucción de la línea 50, **ERASE A**, no podemos dimensionar de nuevo la variable A en la línea 60, pues se producirá el error. "Redimen-

sioned Array", ya que una misma variable no puede dimensionarse dos veces.

**DEFINT** <lista de variable>

Define las variables especificadas como enteras. (sin parte decimal)

- <lista de variables>: variables que van a ser definidas como enteras. Deben ir separadas por una coma.

```
10 DEFINT A, B, X
20 A = 7.18: B = 16.8: X = 1.6
30 PRINT A; B; X
(RUN)
7 16 1
```

**DEFSNG** <lista de variables>

Define las variables especificadas, en simple precisión. (Se representaran 6 dígitos como máximo).

- <lista de variables>: variables que van a ser definidas en simple precisión. Deben ir separadas por una coma.

```
10 DEFSNG A, B, X
20 A = 128345.6342
30 B = 3256.83716
40 X = 12.684372
50 PRINT A;B;X
(RUN)
128345 3256.83 12.6843
```

**DEFDBL** <lista de variables>

Define las variables especificadas, en doble precisión. (Se representaran 14 dígitos como máximo).

- <lista de variables>: variables que van a ser definidas en doble precisión. Deben ir separadas por coma.

```
10 DEFDBL A, B
20 A = 30/4.3: B = 45/87
30 B = PRINT A; B
(RUN)
6.9767441860465    0.51724137931034
```

**DEFSTR** <lista de variables>

Define las variables especificadas como alfanuméricas.

- <lista de variables>: variables que van a ser definidas como alfanuméricas. Deben ir separadas por coma.

```
10 DEFSTR A, B
20 A = "BASIC"
30 B = "MSX"
40 PRINT A + " " + B
(RUN)
BASIC MSX
```

**NOTA:** En todos los comandos de declaración de caracteres (DEFINT, DEFSNG, DEFDBL, DEFSTR), puede especificarse también un rango de variables a declarar. Por ejemplo, DEFINT A-D, define las variables A, B, C, D, de tipo entero.

**DEFFN** <nombre><(lista de parametros)> = <definición de la función>

Permite definir una función por el usuario.

- <nombre>: Puede ser el nombre de cualquier variable. Este <nombre>, precedido de **FN**, será el nombre de la función.

- <(lista de parámetros)>: Estará compuesta por todas las variables que existan en la <definición de la función> y serán sustituidas al llamar a dicha función. Las variables deben estar separadas por coma.

● <definición de la función>: Expresión matemática que realiza las operaciones de la función definida, estando limitada a una línea (255 caracteres).

Las variables utilizadas en esta expresión, únicamente sirven para definir la función, no influyendo las variables del programa que tengan el mismo nombre. Si en posteriores llamadas a esta función, se omite el valor de una variable será tomado su último valor.

La llamada a la función se realizará utilizando la siguiente estructura: **FN** <nombre> <(valores de las variables)>.

```
10 DEFFNR (A,B) = (A*B)/2
.
.
60 A = FNR (4,5)
70 PRINT A
(RUN)
10
```

Si se realiza la llamada de una función antes de que esta haya sido definida, se producirá el error: «Undefined user function».

**DEFFN** no puede utilizarse como instrucción directa.

**DEFUSR** <número> = <dirección>

Define la dirección de inicio de una subrutina en lenguaje máquina.

- <número>: número de la subrutina **USR**. Debe ser un entero entre 0 y 9. Si se omite, se asignará por defecto el valor 0. (**DEFUSR 0**)
- <dirección>: Dirección de inicio de la subrutina **USR** especificada.

```
DEFUSR 0 = 53248
```

**USR** <número> <(x)>

Ofrece el resultado obtenido al ejecutar una rutina en lenguaje máquina que empieza en la dirección definida por **DEFUSR**.

- <número>: Identifica la subrutina con un número determinado que debe ser un entero entre 0 y 9. Si se omite, se asignará por defecto el valor 0. (**USR0**).

- **<(x)>**: Parámetro que será utilizado por la subrutina en lenguaje ensamblador.

**KEY** <número>, <"expresión">

Define las teclas de función F1 a F10

- <número>: número de la tecla de función a definir. Debe ser un entero entre 1 y 10.
- <"expresión">: sentencia, comando, función, operación, etc., que se va a definir.  
Puede tener como máximo 15 caracteres.

```
KEY 1, "BASIC MSX"
```

```
KEY 3, "SQR"
```

```
KEY 4, "3.1416"
```

```
A$ = "ORDENADOR": KEY 5, A$: KEY 7, "LPRINT".
```

Para utilizar el contenido de las teclas programadas, ejecute KEY LIST (ver KEY LIST para más detalles).

## TIME

Temporizador interno del sistema. Al inicializar el sistema se pone a cero y va incrementándose en 1 cada vez que VDP genera interrupción (60 veces por segundo). Dividiendo el valor de **TIME** por 60 obtendremos un contador de segundos.

**TIME** es una variable especial del sistema. Automáticamente toma el valor 0 y va incrementándolo, aunque también podemos asignarle un valor inicial por programa. (**TIME** = <valor>)

```
10 LOCATE 17,2: PRINT INT (TIME/60)
```

```
20 GOTO 10
```

#### 4. COMANDOS Y FUNCIONES DE I/O

- READ
- DATA
- RESTORE
- INPUT
- LINE INPUT
- LET
- MID\$( ) = Y\$
- PRINT
- PRINT USING
- LPRINT
- LPRINT USING
- LPOS
- SWAP
- INKEY\$
- INPUT\$
- FRE
- POKE
- PEEK
- VPOKE
- VPEEK
- INP
- OUT
- WAIT
- STICK
- STRIG

**READ** <lista de variables>

Lée los datos correspondientes a una sentencia **DATA**, asignándolos a las variables especificadas en <lista de variables>.

<lista de variables>: Pueden ser numéricas o alfanuméricas y la única condición que deben cumplir es que estén separadas por coma.

```
READ A$, B, C.
```

**READ** se utiliza conjuntamente con **DATA**. Cuando el Ordenador encuentra una sentencia **READ**, buscará el **DATA** correspondiente en cualquier lugar del programa.

```
10 READ A$, B
:
500 DATA PEDRO, 23
```

Cuando el ordenador llega a la línea 10, realizará una operación de lectura en el **DATA** de la línea 500, asignando a la variable **A\$** el valor PEDRO y a **B** el valor 23.

**DATA** <lista de datos>

Constituye un archivo de datos dentro del programa, que en cualquier momento pueden ser leídos por **READ**.

<lista de datos>: Pueden ser números, caracteres ó ambas cosas y deben estar separados por coma. Estos valores serán asignados sucesivamente a las variables de **READ**.

El número de datos contenido en <lista de datos> nunca puede ser menor que el número de variables contenida en <lista de variables> de la sentencia **READ**. De otra forma, se originaría el error "Out of Data".

Si dentro de <lista de datos> queremos introducir un dato que contenga signos especiales como coma (,) dos puntos (:)... etc., éstos deberán ir entre comillas. Las comillas no pueden utilizarse como dato dentro de **DATA**.

```

10 REM$ EJEMPLO READ/DATA$
20 FOR I = 1 TO 2
30 READ A$, B$, C$
40 NEXT I
50 DATA HIT, BIT, 55, SONY
60 DATA BASIC, MSX

```

En la primera operación de lectura ( $I = 1$ ), los valores asignados a A\$, B\$ y C\$ son respectivamente: HIT, BIT, 55.

En la segunda lectura ( $I = 2$ ), los valores asignados serán: SONY, BASIC, MSX.

#### **RESTORE** <núm. línea>

Indica el número de línea donde se encuentra el **DATA** que contiene los datos a leer por **READ**.

Si no se indica <núm. línea>, se reestablecerá el orden de lectura, colocando el puntero en el primer dato de la primera línea DATA del programa.

```

10 READ A$, B
:
:
50 RESTORE
:
60 READ A$, B
:
:
100 DATA PEDRO, 23

```

Sin la instrucción **RESTORE** de la línea 50, se produciría el error "OUT OF DATA", ya que el **READ** de la línea 60, intentaría leer los datos posteriores a PEDRO, 23 que no existen. De esta forma, restablecemos el orden de lectura, colocando el puntero en el primer dato (PEDRO), por lo que las instrucciones de las líneas 10 y 60, leerán los mismos datos.

```
10 READ A$, B
.
50 RESTORE 520
60 READ C,D,E,F
.
.
500 DATA PEDRO, 23
510 DATA 100, 200
520 DATA 400, 350, 600
530 DATA 800, 900
```

La línea 10, lee los datos PEDRO, 23.

La línea 50, hace que el próximo **READ** se efectúe a partir de la línea 520, originándose los siguientes valores: C = 400, D = 350, E = 600, F = 800.

Sin la instrucción de la línea 50 (**RESTORE 520**), los valores se habrían asignado de la forma:

C = 100, D = 200, E = 400, F = 350.

**INPUT** <lista de variables>

Asigna un valor, introducido desde el teclado, a una variable.

- <lista de variables> Variables (numéricas o alfanuméricas) a las que serán asignados los valores introducidos desde el teclado.

Si durante la ejecución del programa éste se encuentra con el comando **INPUT**, la ejecución se detendrá, apareciendo en pantalla un interrogante (?), y permanecerá a la espera hasta que introduzcamos un dato desde el teclado, que será asignado a la variable asociada al comando **INPUT**.

```
INPUT A
INPUT A$
```

El comando **INPUT** nos permitirá, si así lo deseamos, incluir un mensaje, que deberá ser escrito entre comillas, y que en el momento de ser ejecutado se imprimirá en la pantalla.

```
20 INPUT "ESCRIBE UN NUMERO: ";X
30 PRINT "EL CUADRADO DE"; X; "ES "; X*X
```

Al ejecutar el programa anterior, el ordenador nos pedirá el valor de X:

```
ESCRIBE UN NUMERO: ?
```

Una vez introducido, se lo asignará a la variable X y continuará con la ejecución del programa imprimiendo el cuadrado del valor introducido.

Con un solo comando **INPUT**, pueden asignarse varios valores a distintas variables, siempre que estén separadas por coma: **INPUT A,B,C**. Así mismo, al introducir los datos desde el teclado, deberemos separarlos por coma.

```
10 INPUT "INTRODUZCA LA FECHA"; D,M,A
(RUN)
? 31, 12, 84
```

Si en un comando **INPUT**, que tiene asociada una variable numérica, introducimos un valor alfanumérico, se imprimirá el mensaje de error: «Redo from start», permaneciendo de nuevo a la espera del dato. (Recuerde que después de teclear el valor de entrada debe pulsar la tecla RETURN).

```
10 INPUT "A y B"; A,B
20 PRINT A*B
(RUN)
A y B? 10, PE
REDO FROM START
A y B? 10, 20
200
OK
```

```
10 INPUT "SU NOMBRE"; NOS
20 PRINT "HOLA,"; NOS
```

```
10 INPUT "NOMBRE Y EDAD:" N$,E
20 PRINT, N$; "TIENE "; E; "AÑOS "
```

### **LINE INPUT** <variable>

Asigna una cadena de caracteres, introducidos desde el teclado, a una variable alfanumérica. El número máximo de caracteres de la cadena es 254.

- <variable>: variable alfanumérica a la que se asignará la cadena de caracteres introducida desde el teclado.

**LINE INPUT** es similar a **INPUT**, teniendo en cuenta las siguientes diferencias:

- No imprime el interrogante.
- Sólo puede utilizarse con variables alfanuméricas
- Sólo puede utilizarse con una variable.

```
10 LINE INPUT "NOMBRE, APELLIDOS Y
DIRECCION: "; A$
(RUN)
NOMBRE, APELLIDOS Y DIRECCION:
```

Todos los caracteres, introducidos hasta que se pulse **RETURN**, serán asignados a la variable **A\$**.

**LET** <variable> = <expresión>  
Asigna un valor a una variable.

- <variables>: variable a la que se asignará el valor <expresión>. Puede ser numérica o alfanumérica.
- <expresión>: valor que se asignará a la variable especificada.

```
LET A = 50
LET A$ = "NOMBRE"
LET C = A+B
```

**LET** es opcional, ya que directamente, puede asignarse una expresión a una variable mediante el signo «igual» (=), sin utilizar dicho comando.

Así, las siguientes instrucciones realizan la misma función:

```
10 LET A = 100
```

```
10 A = 100
```

**MID\$** <(X\$, N, M)> = Y\$

Extrae M caracteres de la variable X\$, a partir de la posición. N, sustituyéndolos por M caracteres de la variable Y\$.

Si se omite M, se colocarán todos los caracteres de Y\$.  
Y\$ no debe ser mayor que X\$.

```
10 A$ = "ORDENADOR"
```

```
20 B$ = "SONY "
```

```
30 MID$ (A$, 3,3) = B$
```

```
40 PRINT A$
```

(RUN)

```
ORSONADOR
```

**PRINT** <expresión>

Imprime en pantalla la <expresión>

● <expresión>: Puede ser de varios tipos:

1. Valores numéricos:

```
PRINT 50
```

2. Variables numéricas cuyos valores han sido asignados previamente:

```
10 A = 3
```

```
20 PRINT A
```

Imprime en pantalla: 3

3. Variables de cadena cuyos valores han sido asignados previamente:

```
10 A$ = "MSX"
```

```
20 PRINT A$
```

Imprime en pantalla: MSX

4. Expresiones que contienen operaciones aritméticas o lógicas:

```
10 A = 10
```

```
20 PRINT 3*2+A
```

Imprime en pantalla: 26 (resultado de 3\*2+10)

5. Cadenas encerradas entre comillas:

```
10 PRINT "BASIC MSX"
```

Imprime en pantalla: BASIC MSX

Las expresiones separadas por coma se imprimirán en dos campos distintos:

```
10 A = 5: B = 10
20 PRINT A,B
(RUN)
5 10
```

Las expresiones separadas por punto y coma (;) se imprimirán a continuación de la anterior, teniendo en cuenta que las expresiones numéricas estarán separadas por dos espacios.

Si al final de la última expresión de un comando **PRINT** hay un punto y coma, la próxima sentencia **PRINT** se imprimirá a continuación de ésta.

Si se utiliza **PRINT** sin ninguna expresión, se imprimirá una línea en blanco.

El símbolo de interrogación (?) puede utilizarse en lugar de la palabra **PRINT**:

```
PRINT 10
? 10
```

Ambas expresiones realizan la misma función. (imprimen en pantalla: 10).

**PRINT USING** <Formato>; <expresión>

- Imprime en pantalla la <expresión> utilizando el <formato> especificado.
- <Formato>: caracteres que definen el formato de impresión.
- <expresión>: variables o expresiones que van a ser impresas con el formato especializado. Deberán estar separadas por coma.

Cuando se usa **PRINT USING** para imprimir variables alfanuméricas, pueden utilizarse los siguientes caracteres, para su formateado:

1. "!" indica que sólo se imprimirá el primer carácter de la variable alfanumérica.

2. "+": el signo de adición indica que el signo del número a escribir se colocará delante o detrás del mismo.

```
PRINT USING "+ # # #. # #"; 1.26, -1.25
```

```
+1.25 -1.25.
```

```
Ok
```

```
PRINT USING "# # # . # # +"; 1.25,-1.25
```

```
(RUN)
```

```
1.25 + 1.25-
```

3. "-": el signo de sustracción al final del formato, implica la colocación de dicho signo al final de los números negativos.

```
PRINT USING "# # #. # #-"; 1.25, -1.25
```

```
Ok
```

```
1.25 1.25-
```

4. "\*\* \*": un doble asterisco al principio del formato indica que aparecerá el asterisco en los lugares en que sobre espacio, previos al dígito. También indica dos o más lugares para dígitos.

```
PRINT USING "*** # # . #"; 1.25, -1.25
```

```
*1.25 *-1.25
```

```
Ok
```

5. "££" aparecerá el símbolo de la libra inmediatamente a la izquierda del número en cuestión. Los ££ ofrecen dos posiciones más de dígitos, una de las cuales es el signo de la libra. La forma exponencial no puede ser usada con ££.

```
PRINT USING "££ # # #"; 5680
```

```
£ 5680
```

```
Ok
```

```
A$ = "España"  
Ok  
PRINT USING "!"; A$  
E  
Ok
```

2. "\ n espacios \"", indica que van a imprimirse 2+n caracteres de la variable alfanumérica.

```
A$ = "España"  
Ok  
PRINT USING "\ \ "; A$  
ESPAN  
Ok
```

3. "&" Indica que serán impresos todos los caracteres de la variable.

```
A$ = "España"  
Ok  
PRINT USING "Me gusta mucho &"; A$  
Me gusta mucho España  
Ok
```

Cuando se usa **PRINT USING** para imprimir datos numéricos, pueden utilizarse los siguientes caracteres para su formateado:

1. "#" se usa para representar cada posición de dígito. Si el número a ser impreso tiene menos dígitos que posiciones especificadas, el número será precedido por tantos espacios en blanco como # sobrantes.

```
PRINT USING "#####.##"; 10.983, 8.6, 764.3  
(RUN)  
10.98 8.60 764.30
```

6. "\*\*\*£": combina los efectos de los dos símbolos anteriores. Los espacios en blanco serán llenados con asteriscos y el signo de la libra aparece antes del número. Los \*\*£ ofrecen tres posiciones más de dígitos, uno de los cuales es £.

```
PRINT USING "***£#.# #"; 12.35
* £ 12.35
Ok
```

7. ",": una coma situada a la izquierda del punto decimal en la secuencia implica que una vez ejecutada la sentencia, aparecerá una coma situada tres dígitos a la izquierda del decimal. Una coma situada al final de la secuencia, se imprimirá al final del número.

```
PRINT USING "# # # #, # #"; 1234.5
1,234.50
Ok
```

```
PRINT USING "# #-# #.# #,"; 1234.5
1234.50,
Ok
```

8. "#####": convierten el número dado en exponencial.

```
PRINT USING "# #.# # ^^^^"; 234.56
235 E+02
```

Ok

```
PRINT USING "#.# # # ^^^^-"; -12.34
1.23E-01-
```

Ok

```
PRINT USING "+#.# # ^^^^"; 12.34, -12.34
+1.23E+01 - 1.23E-01
```

Ok

9. "%": si el número a imprimir tiene más dígitos que los previstos en la instrucción, se imprimirá el símbolo % junto al número. Lo mismo

sucede si la operación de redondeo causa que el número contenga más dígitos de los previstos.

```
PRINT USING "# #.# #"; 123.45
```

```
% 123.45
```

```
Ok
```

```
PRINT USING ".# #";.998
```

```
% 100
```

```
OK
```

Si el número de dígitos excede de 24, aparece el mensaje de error «Illegal function call».

### **LPRINT**

Imprime en impresora. Sigue las mismas normas que **PRINT**. (ver **PRINT** para más detalles).

### **LPRINT USING**

Imprime en impresora con un formato específico. Sigue las mismas normas que **PRINT USING**. (ver **PRINT USING** para más detalles).

### **LPOS**

Realiza la función de **POS** referida a la impresora (Ver comando **POS** para más detalles).

### **SWAP <X>, <Y>**

Intercambia los valores de las variables especificadas. Ambas variables deben ser del mismo tipo (enteras, simple precisión,...) de lo contrario se producirá el error: "Type mismatch."

- <X>: variable que tomará el valor de la variable <Y>.
- <Y>: variable que tomará el valor de la variable <X>.

```
10 A = 5: B = 10
```

```
20 PRINT "A = "; A, "B = "; B
```

```
30 SWAP A,B
```

```
40 PRINT "A = "; A, "B = "; B
```

```
(RUN)
```

```
A = 5    B = 10
```

```
A = 10   B = 5
```

### INKEY \$

Devuelve una cadena de un sólo carácter definido por la pulsación de una determinada tecla. Si no se pulsa ninguna tecla durante el periodo de observación, devuelve una cadena nula.

Debido a la corta duración de los ciclos de observación, **INKEY \$** se sitúa dentro de alguna clase de bucle para que sea observado continuamente.

```
10 A$ = INKEY$
```

```
20 IF A$ = " " GOTO 10
```

### INPUT\$ (x)

Detiene el programa hasta que se hayan introducido x caracteres. Los caracteres introducidos no serán visualizados en pantalla. Pulsando **CTRL-C** ó **CTRL-STOP** se anulará **INPUT\$ (x)**.

```
10 X$ = INPUT$(25)
```

### FRE <(expresión)>

Determina el número de **BYTES** libres en memoria **RAM**.

- <(expresión)>: Puede ser cualquier carácter alfanumérico, no influyendo en el resultado. Si el carácter va entre comillas, nos indicará el número de **BYTES** reservados para las cadenas relativas a variables alfanuméricas. (Ver **CLEAR** para más detalles).

```
PRINT FRE (X)
```

```
PRINT FRE ("Z")
```

**POKE** <dirección>, <dato>

Escribe un dato en la posición de memoria especificada.

- <dirección>: Dirección de memoria donde se escribirá el dato. Debe estar comprendida entre -32768 y 65535. Si el valor es negativo, la dirección real será 65.536 menos el valor especificado. (-1, será calculado como  $65.536 - 1 = 65.535$ .) (ver Mapa de memoria)
- <dato>: Dato (**BYTE**) que va a ser escrito en la dirección de memoria especificada. Debe ser un número entre 0 y 255.

```
POKE 63.522, 65
```

**PEEK** <(dirección)>

Es la función complementaria de **POKE**. Lee el dato (número entero entre 0 y 255) almacenado en la dirección de memoria especificada.

- <(dirección)>: Dirección de memoria donde se encuentra el dato que deseamos leer.

Para visualizar el dato leído, utilizaremos PRINT.

```
PRINT PEEK (63.522)
```

**VPOKE** <dirección>, <dato>

Almacena un dato en la dirección especificada de la V RAM. (memoria de pantalla).

- <dirección>: Debe ser un número entero entre 0 y 16.383. (ver Mapa de memoria).
- <dato>: Dato (**BYTE**) que va a ser almacenado en la dirección de memoria especificada. Debe ser un entero entre 0 y 255.

```
VPOKE 14500, 65
```

**VPEEK** <(dirección)>

Es la función complementaria de **VPOKE**. Lee el dato (número entero

entre 0 y 255) almacenado en la dirección especificada de la **VRAM** (memoria de pantalla).

- <dirección>: Debe ser un número entero entre 0 y 16.383.

PRINT VPEEK (14500)

**INP** <port>

Lée el **BYTE** almacenado en el port especificado.

- <port>: número del port donde va a efectuarse la lectura. Debe ser un entero entre 0 y 255.

**INP** es la función complementaria de **OUT**

INP (&H92)

**OUT** <port>, <valor>

Escribe un **BYTE** en el port de salida especificado.

- <port>: número del port donde va a efectuarse la operación de escritura. Debe ser un entero entre 0 y 255.
- <valor>: Dato (**BYTE**) que va a ser transmitido. Debe ser un entero entre 0 y 255.

OUT &H90, 3

**WAIT** <port>, Y, Z

Efectúa una pausa hasta que el dato del port de entrada cambie a un valor determinado.

- <port>: número del port de entrada utilizado.

La secuencia de operaciones que implica la ejecución de esta instrucción, son las siguientes:

1. Lée el dato en <port>
2. Opera en forma OR exclusiva (EX-OR) el dato leído con el valor Z.

3. Efectúa el producto lógico AND entre el resultado de la operación EX-OR y el valor Y.

Si el resultado es 0, realiza de nuevo todo el proceso. En caso contrario prosigue la ejecución del programa a partir de la instrucción que sigue a **WAIT**.

### **STICK <(N)>**

Determina el movimiento realizado por el Joystick, dando un valor numérico según la siguiente codificación:

Movimiento	STICK (N)
● Centro	0
● Ascendente	1
● Diagonal derecha ascendente	2
● Derecha	3
● Diagonal derecha descendente	4
● Descendente	5
● Diagonal izquierda descendente	6
● Izquierda	7
● Diagonal izquierda ascendente	8

●<(N)>: indica el Joystick utilizado.  
N = 0: teclado  
N = 1: Joystick 1  
N = 2: Joystick 2

Conecte el Joystick 1 y mueva la palanca para visualizar los distintos valores tomados según el movimiento realizado:

```
10 LOCATE 5,3
20 PRINT STICK (1)
30 GOTO 10
```

### **STRIG <(N)>**

Toma el valor -1 al pulsar el botón de disparo del Joystick. Si no se pulsa, toma el valor 0.

●<(N)>: Indica el Joystick utilizado:

N = 0: teclado (botón de disparo = barra espaciadora)

N = 1: Joystick 1

N = 2: Joystick 2

## 5. INTERRUPCIONES

- **ON KEY GOSUB**
- **KEY (N) ON/OFF/STOP**
- **ON STRIG GOSUB**
- **STRIG (N) ON/OFF/STOP**
- **ON STRIG GOSUB**
- **STOP ON/OFF/STOP**
- **ON SPRITE GOSUB**
- **SPRITE ON/OFF/STOP**
- **ON INTERVAL = < > GOSUB**
- **INTERVAL ON/OFF/STOP**

**ON KEY GOSUB** <nº de línea>

Interrumpe el programa principal al pulsar una determinada tecla de función (F1-F10), pasando a realizar la subrutina de interrupción indicada en <nº de línea>.

● <nº de línea>: números de líneas iniciales de las subrutinas.

ON KEY GOSUB 100, 200, 350, 760,...

100 es la línea de inicio de la subrutina de interrupción correspondiente a la tecla de función F1. **(KEY (1))**.

200 es la línea de inicio de la subrutina de interrupción correspondiente a la tecla de función F2. (**KEY (2)**).

•  
•  
•  
•

Al producirse una interrupción, se ejecutará automáticamente un **KEY (n) STOP**. (ver comando **KEY ON/OFF/STOP**).

Si dentro de la subrutina de interrupción no se ha especificado **KEY-(n)OFF**, después de ser ejecutado el **RETURN**, (retorno de la subrutina al programa principal) se ejecutará también un **KEY(n)ON**, quedando de nuevo habilitada la interrupción para esta tecla.

Si se produce una interrupción de error (debida a un comando **ON ERROR**), se inhibirán automáticamente todas las interrupciones. (incluidas **ERROR, STRIG, STOP, SPRITE, INTERVAL, KEY**).

### **KEY ON/OFF/STOP**

Se utilizan para habilitar/inhibir interrupciones con las teclas de función Fn. (**KEY(n)**).

- **KEY(n)ON**: Permite interrupción con la tecla de función Fn. (**KEY(n)**).

```
10 KEY (1) ON
20 ON KEY GOSUB 200
30 REM PROGRAMA PRINCIPAL
40 FOR X = 1 TO 100
50 PRINT X
60 NEXT X
70 CLS
80 GOTO 30
200 REM SUBROUTINA DE INTERRUPCION
210 CLS.
230 PRINT "HAS PULSADO F1"
230 FOR I=1 TO 100
```

```
240 LOCATE 0,10: PRINT I
250 NEXT I
260 CLS
270 RETURN
```

En la línea 10, indicamos al ordenador que «permitimos» interrumpir el programa pulsado F1.

En la línea 20, le indicamos que si durante el transcurso del programa principal se pulsa la tecla F1, pase a ejecutar la subrutina que empieza en la línea 200.

El programa principal (10-80) se ejecutará con toda normalidad hasta que pulsemos F1, momento en que se detendrá, pasando a ejecutar la subrutina de interrupción correspondiente. Una vez finalizada la ejecución de esta subrutina el programa seguirá su curso a partir del punto donde fue interrumpido, pudiendo ser interrumpido de nuevo del mismo modo anterior.

**KEY(n)OFF:** Inhibe la interrupción de la tecla de función Fn. (KEY(n)). Si en el ejemplo anterior sustituimos la línea 10 por: 10 KEY (1) OFF, aunque pulsemos F1, no se interrumpirá el programa siguiendo éste su curso normal.

**KEY(n)STOP:** Si previamente a KEY(n)STOP se habilitó la interrupción mediante KEY(n)ON, al pulsar la tecla Fn no se producirá interrupción, pero memorizará que se pulsó dicha tecla y pasará a ejecutar la subrutina correspondiente después de encontrar la autorización de interrupción KEY(n)ON.

```
10 KEY(1)ON
20 PRINT "BASIC MSX"
30 KEY(1) STOP
40 ON KEY GOSUB 500
.
.
.
100 PRINT "FIN = F1"
```

```
110 KEY(1)ON
```

```
500 PRINT "FIN"
```

```
510 RETURN
```

Aunque pulsemos F1, el programa no será interrumpido hasta que aparezca el mensaje que imprime la línea 100: «FIN = F1», ya que es en la línea 110 donde se habilitó de nuevo la interrupción para F1.

### **ON STRIG GOSUB** <nº de líneas>

Interrumpe el programa principal cuando se pulsa el botón de disparo de los Joysticks o la barra espaciadora del teclado.

Su funcionamiento es similar a **ON KEY GOSUB**. (Ver **ON KEY GOSUB** para más detalles).

### **STRIG(n) ON/OFF/STOP**

Se utilizan para habilitar/anular interrupciones con el botón de disparo de los Joysticks o con la barra espaciadora del teclado.

(n) debe ser un número entero entre 0 y 4.

- n = 0 : la barra espaciadora es utilizada como botón de disparo.
- n = 1 ó 3: se utiliza el botón de disparo de Joystick 1.
- n = 2 ó 4: se utiliza el botón de disparo del Joystick 2.

Su funcionamiento es similar a **KEY ON/OFF/STOP** (Ver **KEY ON/OFF/STOP** para más detalles).

### **ON STOP GOSUB** <nº. de línea>

Interrumpe el programa principal al pulsar **CTRL-STOP**, pasando a realizar la subrutina de interrupción cuya línea de inicio es la indicada en <nº. de línea>.

Utilice con precaución este comando. Por ejemplo, el siguiente programa no podrá detenerse. La única solución es hacer un **RESET** ó desconectar el aparato:

```
10 ON STOP GOSUB 40
```

```
20 STOP ON
```

```
30 GOTO 30
```

```
40 RETURN
```

### **STOP ON/OFF/STOP**

Se utilizan para habilitar/anular interrupciones mediante **CTRL-STOP**. Su funcionamiento es similar a **KEY ON/OFF/STOP**. (ver **KEY/ON/OFF/STOP**).

### **ON SPRITE GOSUB <n° línea>**

Interrumpe el programa principal cuando se solapan dos **SPRITES** en la pantalla.

Su funcionamiento es similar a **ON KEY GOSUB** (ver **ON KEY GOSUB** para más detalles).

### **SPRITE ON/OFF/STOP**

Se utilizan para habilitar/anular interrupciones cuando coinciden dos **SPRITES**.

Su funcionamiento es similar a **KEY ON/OFF/STOP**. (ver **KEY ON/OFF/STOP** para más detalles).

### **ON INTERVAL =<tiempo> GOSUB <n° línea>**

Interrumpe el programa principal cada periodo de tiempo especificado, en  $\langle \text{tiempo} \rangle / 60$ .

```
10 ON INTERVAL = 60 GOSUB 500
```

Al ejecutar esta línea, cada segundo (60/60) se interrumpirá el programa principal para ejecutar la subrutina que comienza en la línea 500.

Su funcionamiento es similar a **ON KEY GOSUB**, con la diferencia de que la interrupción se producirá automáticamente cada cierto tiempo, sin tener que pulsar ninguna tecla.

### **INTERVAL ON/OFF/STOP**

Se utiliza para habilitar/anular interrupciones temporales con **INTERVAL**.

Su funcionamiento es similar a **KEY ON/OFF/STOP**.

(Ver **KEY ON/OFF/STOP** para más detalles).

### STOP ON OFF STOP

Se utiliza para definir un intervalo de tiempo entre un STOP y el siguiente STOP. El funcionamiento es similar a KEY ON OFF STOP. (Ver KEY ON OFF STOP)

### ON SPRITE GOSUB -> GOSUB

Interrumpe el programa principal cada vez que se ejecuta una línea de código que comienza con ON SPRITE GOSUB. El funcionamiento es similar a ON KEY GOSUB (ver ON KEY GOSUB para más detalles).

### SPRITE ON OFF STOP

Se utiliza para definir un intervalo de tiempo entre un SPRITE y el siguiente SPRITE. El funcionamiento es similar a KEY ON OFF STOP. (Ver KEY ON OFF STOP para más detalles).

### FOR INTERVAL -> GOSUB -> GOSUB

Interrumpe el programa principal cada vez que se ejecuta una línea de código que comienza con FOR INTERVAL. El funcionamiento es similar a ON KEY GOSUB (ver ON KEY GOSUB para más detalles).

Se utiliza para definir un intervalo de tiempo entre un FOR INTERVAL y el siguiente FOR INTERVAL. El funcionamiento es similar a ON KEY GOSUB (ver ON KEY GOSUB para más detalles).

Al ejecutar este ítem, este programa (PROM) se interrumpirá si se ejecuta una línea de código que comienza con ON KEY GOSUB. El funcionamiento es similar a ON KEY GOSUB con la diferencia de que la interrupción se produce solo cuando se ejecuta una línea de código que comienza con ON KEY GOSUB.

### INTERVAL ON OFF STOP

Se utiliza para definir un intervalo de tiempo entre un INTERVAL ON OFF STOP y el siguiente INTERVAL ON OFF STOP. El funcionamiento es similar a ON KEY GOSUB (ver ON KEY GOSUB para más detalles).

### ON KEY ON OFF STOP

Se utiliza para definir un intervalo de tiempo entre un ON KEY ON OFF STOP y el siguiente ON KEY ON OFF STOP. El funcionamiento es similar a ON KEY GOSUB (ver ON KEY GOSUB para más detalles).

## 6. FUNCIONES NUMERICAS

- ABS
- INT
- FIX
- SGN
- CDBL
- CSNG
- CINT
- EXP
- LOG
- SQR
- SIN
- COS
- TAN
- ATN
- RND

### ABS (X)

Da el valor absoluto de X. El resultado siempre será positivo.

```
10 PRINT ABS (-5)
```

```
(RUN)
```

```
5
```

## INT (X)

Calcula el mayor entero menor o igual a X.

```
10 PRINT INT (2.56)
```

```
(RUN)
```

```
2
```

```
10 PRINT INT (-2.56)
```

```
(RUN)
```

```
-3
```

## FIX (X)

Da la parte entera de X. **FIX (X)** es equivalente a  $\text{SGN}(X) * \text{INT}(\text{ABS}(X))$ .

La mayor diferencia entre **FIX** e **INT** es que cuando trabajan con números negativos ( $X < 0$ ) **FIX** da la parte entera por exceso e **INT** por defecto.

```
10 PRINT FIX (-2.56)
```

```
(RUN)
```

```
-2
```

## SGN (X)

Determina el signo de X:

Si  $X > 0$  :  $\text{SGN}(X) = 1$

Si  $X < 0$  :  $\text{SGN}(X) = -1$

Si  $X = 0$  :  $\text{SGN}(X) = 0$

## CDBL (X)

Representa el número X en doble precisión.

```
10 A = 53.45567893141785
```

```
20 PRINT CDBL (A)
```

```
(RUN)
```

```
53.455678931418
```

**CSNG (X)**

Representa el número X en simple precisión.

```
10 A = 45.987666666634
20 PRINT CSNG (A)
(RUN)
53.4457
```

**CINT (X)**

Representa el número X como entero, eliminando la parte decimal. X debe estar comprendido entre  $-32768$  y  $32767$ . En caso contrario se producirá error de Overflow (desbordamiento)

```
10 A = 45.98765
20 PRINT CINT (A)
(RUN)
45
```

**EXP (X)**

Calcula la expresión  $e^x$ . X debe ser menor o igual a  $145.06286085862$  (e, es la base de los logaritmos naturales  $e = 2,7182818284588$ ).

```
10 PRINT EXP (2)
(RUN)
7.38905609893
```

**LOG (X)**

Calcula el logaritmo neperiano de X. (logaritmo en base e).

Para obtener el logaritmo de X en otra base (base Y) deberá realizar la siguiente operación:

$\text{LOG (X)/LOG (Y)} = \text{Logaritmo de X en base Y.}$

1. logaritmo neperiano de 4:

```
10 PRINT LOG (4)
```

```
(RUN)
```

```
1.3862943611199
```

2. logaritmo de 1000 en base 10

```
10 PRINT LOG (1000)/LOG (10)
```

```
(RUN)
```

```
3
```

### **SQR (X)**

Calcula la raíz cuadrada de X. X debe ser un número mayor o igual a cero. De lo contrario se producirá el error: "Illegal function call."

```
10 PRINT SQR (25)
```

```
(RUN)
```

```
5
```

### **SIN (X)**

Da el valor de seno de X. (en radianes).  
El cálculo se efectúa en doble precisión.

```
10 PRINT SIN (3.1416/2)
```

```
(RUN)
```

```
.99999999999324
```

### **COS (X)**

Da el valor de coseno de X. (en radianes).  
El cálculo se efectúa en doble precisión.

```
10 PRINT COS (0)
```

```
(RUN)
```

```
1
```

### TAN (X)

Da el valor de tangente de X (en radianes).  
El cálculo se efectúa en doble precisión.

```
10 PRINT TAN (0.1)
      (RUN)
      .10033467208545
```

### ATN (X)

Da el ángulo (en radianes), cuya tangente es X. El resultado estará comprendido entre  $-\pi/2$  y  $+\pi/2$ . El cálculo se efectúa siempre en doble precisión.

```
10 PRINT ATN (0.3)
      (RUN)
      .29145679447789
```

### RND (X)

Genera un número aleatorio entre 0 y 1.

El argumento de **RND**, X, se utiliza para controlar la generación del número aleatorio. Cada vez que se ejecute un programa que contenga **RND**, se generará la misma secuencia de números aleatorios. Durante la ejecución de un programa, si  $X < 0$  se generará la misma secuencia anterior. Si  $X = 0$  repetirá el último número generado y si  $X > 0$ , generará el próximo número aleatorio de la secuencia.

```
10 PRINT RND (1)*100
20 PRINT RND (-1)*100
30 PRINT RND (0)*100
40 PRINT RND (1)*100
      (RUN)
59.521943994623
4.384820420821
4.384820420821
9.62486816692
```

TAM (X)

Da el valor de tangente de X en radianes.  
El cálculo es exacto en todas precisiones.

TO REPT TAM (X) 11  
FREN  
10000000000

ATN (X)

Da el ángulo (en radianes) cuyo tangente es X. El ángulo está en el intervalo  $-\pi/2 < X < \pi/2$ . El cálculo es exacto en todas precisiones.

TO REPT ATN (X) 11  
FREN  
20000000000

RND (X)

Genera un número aleatorio entre 0 y 1.  
El argumento de RND, X, se utiliza para controlar la generación del número aleatorio. Cada vez que se ejecuta un programa que contenga RND, se genera la misma secuencia de números aleatorios. Cuando se ejecuta el programa de nuevo se genera la misma secuencia. El número de X = 0 genera el mismo número aleatorio y X = 1 genera el mismo número aleatorio.

TO REPT RND (X) 11  
FREN  
10000000000  
20000000000  
30000000000  
40000000000  
50000000000  
60000000000  
70000000000  
80000000000  
90000000000  
100000000000

## 7. CONVERSION DE CODIGOS

- **ASC**
- **CHR\$**
- **BIN\$**
- **OCT\$**
- **HEX\$**
- **VAL**
- **STR\$**

**ASC <(X\$)>**

Determina el código ASCII del carácter X\$.

```
10 A$ = "A"  
20 PRINT ASC (A$)  
30 PRINT ASC ("B")  
(RUN)  
65  
66
```

**CHR \$ (X)**

Realiza la función inversa de ASC. Determina el carácter que en código ASCII corresponde al número X.

```
10 PRINT CHR $ (65)
(RUN)
A
```

#### **BINS (X)**

Convierte el número decimal X en un número binario. El valor de X, debe estar comprendido entre  $-32768$  y  $65535$ .

Si X es negativo, dará el valor resultante de restar a  $65.535$  el valor de X. ( $65.535 - X$ ).

```
10 PRINT BINS$ (67)
(RUN)
1000011
```

#### **OCT\$ (X)**

Convierte el número decimal X en un número octal. El valor de X, debe estar comprendido entre  $-32768$  y  $65535$ .

Si X es negativo, dará el valor resultante de restar a  $65.535$  el valor de X. ( $65535 - X$ ).

```
10 PRINT OCT$ (76)
(RUN)
114
```

#### **HEX\$ (X)**

Convierte el número decimal X en un número hexadecimal. El valor de X, debe estar comprendido entre  $-32768$  y  $65.535$ .

Si X es negativo, dará el valor resultante de restar a  $65.535$  el valor de X. ( $65535 - X$ ).

```
10 PRINT HEX$ (94)
(RUN)
5E
```

### VAL (X\$)

Determina el valor numérico de los primeros caracteres numéricos de X\$.

```
10 A$ = "125A43"  
20 B$ = "15C"  
30 PRINT VAL (A$)  
40 PRINT VAL (A$)+VAL (B$)  
50 PRINT VAL (A$+B$)  
(RUN)  
125  
140  
12515
```

### STR\$ (X)

Realiza la función inversa de VAL (X\$).

```
10 A$ = STR$ (22)  
20 B$ = "BASIC"  
30 PRINT A$ + B$.  
(RUN)  
22 BASIC
```

VAL (X)

Determine el valor numérico de los siguientes cálculos utilizando la  
X

1000 - 1000 = 0  
2000 - 1000 = 1000  
3000 - 1000 = 2000  
4000 - 1000 = 3000  
5000 - 1000 = 4000  
6000 - 1000 = 5000  
7000 - 1000 = 6000  
8000 - 1000 = 7000  
9000 - 1000 = 8000  
10000 - 1000 = 9000

VAL (X)

Realice la función inversa de VAL (X)

1000 - 1000 = 0  
2000 - 1000 = 1000  
3000 - 1000 = 2000  
4000 - 1000 = 3000  
5000 - 1000 = 4000  
6000 - 1000 = 5000  
7000 - 1000 = 6000  
8000 - 1000 = 7000  
9000 - 1000 = 8000  
10000 - 1000 = 9000

## 8. CADENAS DE CARACTERES

- **RIGHT\$**
- **LEFT\$**
- **MID\$**
- **STRING\$ (N,X)**
- **STRING\$ (N,X\$)**
- **SPACES\$**
- **INSTR**
- **LEN**

**RIGHT\$<(X\$,N)>**

Extrae N caracteres de la cadena X\$ empezando por la derecha.

Si  $N = \text{LEN}(X\$)$ , nos dará X\$

```
10 A$ = "BASIC MSX"  
20 PRINT RIGHT$(A$,6)  
30 PRINT RIGHT$(A$,3)  
  
(RUN)  
IC MSX  
MSX
```

**LEFT\$ <(X\$,N)>**

Extrae N caracteres de la cadena X\$ empezando por la izquierda.

```
10 A$ = "BASIC MSX"
```

```
20 PRINT LEFT$(A$,7)
```

```
30 B$ = LEFT$(A$,3)
```

```
40 PRINT B$
```

```
(RUN)
```

```
BASIC M
```

```
BAS
```

**MID\$ <(X\$, N, M)>**

Extrae N caracteres de la cadena X\$, empezando por la izquierda, a partir de la posición M.

```
10 A$ = "ORDENADOR SONY"
```

```
20 B$ = MID$(A$, 8, 5)
```

```
30 PRINT B$
```

```
40 PRINT MID$("MSX", 1,2)
```

```
(RUN)
```

```
NADOR SONY
```

```
S
```

**STRING <(N, X)>**

Crea una cadena con N caracteres X, en código ASCII.

```
10 PRINT STRING$(3,65)
```

```
(RUN)
```

```
AAA
```

NOTA: El ASCII de 65 es A.

**STRING\$ <(N, X\$)>**

Crea una cadena con N caracteres X\$ iguales.

```

10 A$ = "A"
20 PRINT STRING$(5,A$)
   (RUN)
   AAAAA

```

**SPACE\$(X)**

Devuelve una cadena de X espacios. El valor de X debe estar comprendido entre 0 y 255.

```

10 A$ = SPACE$(5)
20 PRINT A$

```

**INSTR(N, X\$, Y\$)**

Analiza si la variable Y\$ está contenida en la variable X\$, a partir de la posición N.

X\$ e Y\$, pueden tener un máximo de 255 caracteres.

Si Y\$ no está contenida en X\$, nos dará el valor 0. De lo contrario nos dará el número de la posición a partir del cual Y\$ está contenido en X\$.

```

20 A$ = "ORDENADOR"
20 B$ = "DENA"
30 PRINT INSTR (2,A$,B$)
   (RUN)
   3

```

**LEN(X\$)**

Calcula el número de caracteres que contiene la cadena X\$.

```

10 A$ = "ORDENADOR SONY"
20 PRINT LEN (A$)
   (RUN)
   14
10 A = LEN ("SONY")
20 PRINT A
   (RUN)
   4

```

10-12-2012  
10-12-2012  
10-12-2012

GRABES <Q>  
Cuentas con el signo de X aparecen en el valor de X de las filas con  
cero en Y y Z.

10-12-2012  
10-12-2012

INTER <N, X, Y, Z>  
Analiza si la variable Y está contenida en la variable X, a partir de la  
posición N.  
X e Y pueden tener un número de bits diferente.  
Si Y no está contenida en X, nos dará el valor 0. Si lo contiene por  
determinada posición, nos dará el valor Y en esa posición.

10-12-2012  
10-12-2012  
10-12-2012  
10-12-2012

LEN <X>  
Calcula el número de caracteres que contiene la cadena X.

10-12-2012  
10-12-2012  
10-12-2012  
10-12-2012

## 9. PROCESO DE ERRORES

- **ON ERROR GOTO**
- **ERL/ERR**
- **RESUME**
- **ERROR**

### **ON ERROR GOTO** <nº línea>

Interrumpe el programa principal al producirse un error, pasando a realizar la subrutina de interrupción especificada en <nº línea>.

●<nº línea>): línea inicial de la subrutina de interrupción.

Una vez activada la interrupción, todos los errores detectados, incluidos los errores en modo directo (p.e. "Syntax error") irán a la subrutina de error.

Para desactivar la interrupción de **ERROR**, debe ejecutarse **ON ERROR GOTO 0**. Los siguientes errores imprimirán en pantalla su mensaje de error correspondiente y detendrán la ejecución del programa.

Una instrucción **ON ERROR GOTO 0** que empieza en una subrutina de interrupción de **ERROR** detendría el programa, imprimiendo el mensaje de error que causó la interrupción.

Si se produce un error durante la ejecución de la subrutina, se imprimirá el mensaje correspondiente y se detendrá la ejecución.

Dentro de una subrutina de **ERROR** no puede producirse una interrupción de **ERROR**.

## ERL/ERR

Cuando se produce una interrupción de error, la variable **ERR** contiene el código del error producido (ver códigos de error) y la variable **ERL** contiene el número de línea en el que se originó el error

**ERR** y **ERL** se utilizan normalmente con la instrucción **IF THEN** para dirigir el programa hacia una subrutina de error.

Si la instrucción que motivó el error fue una instrucción en modo directo, **ERL** será igual a 65.535. Para comprobar si se ha producido error en una instrucción directa puede utilizarse: **IF 65535 = ERL THEN...** De lo contrario, puede utilizarse:

**IF ERL = <número de línea> THEN...**

**IF ERR = <código de error> THEN...**

Al ser **ERL** y **ERR** variables reservadas, no pueden aparecer a la izquierda del signo igual (=) en una instrucción **LET**.

## RESUME

### RESUME 0

### RESUME NEXT

### RESUME <núm línea>

Continúa la ejecución de un programa tras haber rectificado el error indicado.

- **RESUME** o **RESUME 0**: La ejecución se reanuda en la instrucción que causó el error.
- **RESUME NEXT**: La ejecución se reanuda en la instrucción siguiente a la que causó el error.
- **RESUME <núm. línea>**: La ejecución se reanuda en el número de línea especificado.

Una instrucción **RESUME** que no esté en la subrutina de error, provoca el error: "RESUME without".

## ERROR <códigos>

Imprime en pantalla el mensaje de error correspondiente al código de error especificado en <códigos>, si éste valor es menor de 60. Si <código> está comprendido entre 60 y 255 se utiliza para la definición de errores por el usuario.

- <código>: Debe ser un número entero comprendido entre 1 y 255, siendo 1 al 59, códigos reservados para el **BASIC** y el resto códigos para definición del usuario.

```
10 ON ERROR GOTO 1000
```

```
120 IF A$ = "N" THEN ERROR 250
```

```
1000 IF ERR = 250 THEN PRINT "ESTAS SEGURO?"
```

Si una instrucción de error especifica un código para el cual no se ha definido ningún mensaje de error se producirá el error: "Unprintable error".

La ejecución de una instrucción de error para la que no hay subrutina de interrupción causará el mismo error anterior.

```
10 ON ERROR GOTO 500
```

```
60 PRINT SQR (-2)
```

```
500 IF ERR = 5 THEN RESUME NEXT
```



## 10. PANTALLA Y GRAFICOS

- SCREEN
- WIDTH
- CLS
- COLOR
- LOCATE
- TAB
- SPC
- POS
- CSRLIN
- KEY ON/OFF
- PSET
- PRESET
- LINE
- CIRCLE
- PAINT
- DRAW
- SPRITE\$
- PUT SPRITE
- POINT
- BASE

**SCREEN** <modo>, <tamaño Sprites>, <sonido teclado>, <velocidad de transmisión>, <impresora>

Define los parámetros especificados.

● <modo>: Selecciona la pantalla para trabajar con texto ó con gráficos.

- 0: Modo texto (40 × 24)
- 1: Modo texto (32 × 24)
- 2: Modo gráfico (256 × 192)
- 3: Modo gráfico (256/4 × 192/4)

Cada punto en modo 3, corresponde a 16 puntos en modo 2. (4 × 4).

● <tamaño Sprites>: Determina el tamaño del Sprite.

- 0: 8 × 8 normal
- 1: 8 × 8 ampliado
- 2: 16 × 16 normal
- 3: 16 × 16 ampliado

● <sonido teclado>: Habilita ó inhibe el sonido producido al pulsar una tecla. ("clik").

- 0: sin sonido
- 1: con sonido

● <velocidad de transmisión>: Determina la velocidad de transmisión de un programa en cassette.

- 1: 1.200 Baudios.
- 2: 2.400 Baudios.

La velocidad de almacenamiento (cassette), también puede determinarse con el comando CSAVE, utilizando la opción <velocidad>.

● <impresoras>: Especifica si la impresora conectada es MSX o no. Si la impresora no es MSX, los símbolos gráficos son convertidos a espacios.

- 0: MSX
- 1: no MSX

### **WIDTH <(X)>**

Proporciona una pantalla con X caracteres por línea en modo texto. (ver comando **SCREEN**)

- <(X)>: número de caracteres por línea. Debe ser un entero comprendido entre 0 y 39 en modo 40 × 24 o entre 0 y 31 en modo 32 × 24.

## **CLS**

Borra el contenido de la pantalla, manteniendo el programa almacenado en memoria.

## **COLOR <X>, <Y>, <Z>**

Define el color de la pantalla.

- <X>: color de los caracteres.
- <Y>: color del fondo.
- <Z>: color del margen.

El valor de <X>, <Y> y <Z>, debe ser un entero comprendido entre 0 y 15.

Si se omite alguno de estos parámetros, mantendrá el valor anterior.

Al inicializar el sistema, los valores asignados son X = 15, Y = 4, X = 7. (COLOR 15,4,7)

El color del margen sólo puede visualizarse en SCREEN 1, SCREEN 2, y SCREEN 3 (Inicialmente el ordenador está en SCREEN 0).

### **Código de colores:**

- 0 — Transparente
- 1 — Negro
- 2 — Verde
- 3 — Verde claro
- 4 — Azul oscuro
- 5 — Azul claro
- 6 — Rojo oscuro
- 7 — Azul celeste
- 8 — Rojo
- 9 — Rojo claro
- 10 — Amarillo oscuro
- 11 — Amarillo claro
- 12 — Verde oscuro
- 13 — Magenta
- 14 — Gris
- 15 — Blanco

## **LOCATE <X>, <Y>**

Situa el cursor en una determinada posición de la pantalla (solo puede utilizarse en SCREEN 0 y SCREEN 1).

- <X>: coordenada horizontal. Debe ser un entero comprendido entre 0 y el valor máximo de la anchura de pantalla menos 1.
- <Y>: coordenada vertical. Debe ser un entero comprendido entre 0 y 22.

**LOCATE** se utiliza normalmente para situar el punto inicial de escritura utilizando los comandos **PRINT** e **INPUT** en la siguiente instrucción del programa.

```
10 LOCATE 10,12: PRINT "LOCATE"
10 LOCATE 5,10: INPUT "NOMBRE:"; A$
```

#### **TAB <(X)>**

Desplaza el cursor (X) espacios en sentido horizontal.

- <(X)>: número de posiciones a desplazar. Debe ser un entero entre 0 y 255.

TAB, sólo puede utilizarse con PRINT y LPRINT.

```
10 PRINT TAB (10) "BASIC MSX"
```

#### **SPC <(X)>**

Imprime X espacios en la pantalla

- <(X)>: número de espacios a imprimir. Debe ser un entero entre 0 y 255.

SPC, sólo puede utilizarse con PRINT y LPRINT.

```
10 PRINT SPC (10)
```

#### **POS <(X)>**

Proporciona la coordenada horizontal actual del cursor.

- <(X)>: Puede ser cualquier caracter, no influyendo en el resultado.

```
10 PRINT POS(A)
10 IF POS (1) = 5 GOTO 200
```

### CSRLIN

Proporciona la coordenada vertical actual del cursor.

```
10 PRINT CSRLIN
10 IF CSRLIN = 15 GOTO 200
```

### KEY ON/OFF

Visualiza o borra el contenido de las teclas de función en la última línea de la pantalla.

**PSET** <(coordenadas)>, <color>

Dibuja un punto en las coordenadas especificadas con el color indicado.

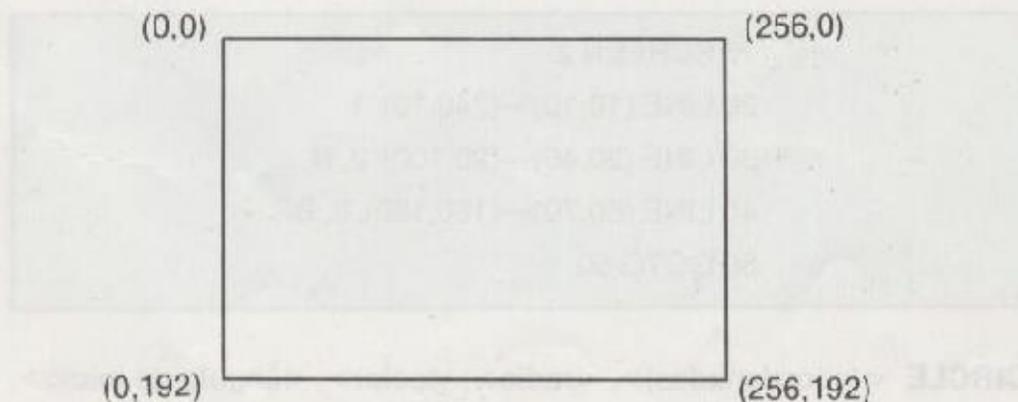
Sólo puede utilizarse en modo gráfico (SCREEN 2/SCREEN 3).

**PRESET** <(coordenadas)>, <color>

Borra un punto de la pantalla en las coordenadas especificadas. Si se especifica el parámetro <color>, actúa igual que PSET.

Sólo puede utilizarse en modo gráfico. (SCREEN 2/SCREEN 3).

**NOTA** En modo gráfico, la pantalla queda dividida en una rejilla de 256 puntos horizontales por 192 puntos verticales:



```

10 SCREEN 2
20 COLOR 1,1,1
30 FOR A = 1 TO 500
40 X = INT (RND(1) * 255)
50 Y = INT (RND(1) * 191)
60 PSET (X,Y), 15
70 PRESET (X+1, Y+1), 2
80 NEXT A
90 GOTO 30

```

**LINE** <(coordenada 1)> - <(coordenada 2)>, <color>

Dibuja una línea entre los dos puntos especificados, con el color indicado.

Sólo puede utilizarse en modo gráfico. (SCREEN 2/SCREEN 3).

- <(coordenada 1)>: coordenadas del punto de inicio.
- <(coordenada 2)>: coordenadas del punto final.
- <color>: color de la línea. (ver código de colores).

Si después de <color> se especifica el parámetro B, dibujará un rectángulo, teniendo en cuenta que los puntos dados como coordenadas deben corresponder a una de las diagonales.

Así mismo, si en lugar de especificar el parámetro B se especifica BF, pintará un rectángulo con el color indicado en <color>.

```

10 SCREEN 2
20 LINE (10,10)—(240,10), 1
30 LINE (20,40)—(90,100), 2, B
40 LINE (60,70)—(160,180), 8, BF
50 GOTO 50

```

**CIRCLE** <(coordenadas)>, <radio>, <color>, <ángulo de inicio>, <ángulo final>, <excentricidad>

Dibuja círculos, elipses y arcos según los parámetros especificados. Los únicos parámetros obligatorios son <coordenadas> y <radio>. (Sólo puede utilizarse en modo gráfico: SCREEN 2/SCREEN 3).

- <(coordenadas)>: coordenadas del punto central
- <radio>: radio
- <color>: color (ver código de colores)
- <ángulo inicial> <ángulo final>: ángulo inicial y final de la figura. Deben expresarse en radianes, entre 0 y  $2\pi$ . Si el ángulo es negativo, la elipse se conectará al punto central con una línea, y el ángulo será tratado como si fuera positivo. (Observe que esto es distinto a sumar  $2\pi$ ).

<excentricidad>: Relación entre los ejes horizontal y vertical.

Si <excentricidad> = 2, el eje horizontal será la mitad del vertical, permitiendo así definir la forma deseada en la elipse.

5 SCREEN 2

10 CIRCLE (125, 96), 70

10 CIRCLE (125, 96), 70, 1, ., 1.2

10 CIRCLE (125, 96), 70, 1, 0, 2\*3.14, 2

10 CIRCLE (125, 96), 70, 1, 0, 2\*3.14, 5

10 CIRCLE (125, 96), 70, 1, ., 0

NOTA Para dibujar un círculo debe especificarse en excentricidad la relación 1.2.

**PAINT** <coordenadas>, <color 1>, <color 2>

Rellena de color un contorno cerrado.

Sólo puede utilizarse en modo gráfico (SCREEN 2/SCREEN 3).

- <coordenadas>: coordenadas del punto desde donde se empezará a pintar.

Si corresponden a un punto interior del contorno cerrado, rellenará de color el interior de dicho contorno, de lo contrario pintará el exterior.

- <color 1>: color con el que se rellenará el contorno (interior o exterior).

- <color 2>: color que limita la superficie a pintar con <color 1>. En alta resolución este parámetro no tiene efecto, debiendo ser el color del contorno a pintar del mismo color que <color 1>.

Si no se especifican los parámetros de color, el color utilizado será el primer parámetro del comando COLOR actual. (color de los caracteres).

```

10 SCREEN 3
20 CIRCLE (125, 96), 50, 1
30 PAINT (125, 96), 2, 1
40 PAINT (10, 10), 10, 1
50 GOTO 50

```

```

10 SCREEN 2
20 CIRCLE (125, 96), 50, 1
30 PAINT (125, 96), 1
40 GOTO 40

```

### **DRAW**

Sentencia de ejecución de los comandos del Macro lenguaje gráfico. (ver Anexo de gráficos).

### **SPRITE\$**

Variable especial del sistema utilizada para la definición de gráficos SPRITE. (Ver Anexo de gráficos).

**PUT SPRITE** <nº plano>, <coordenadas>, <color>

Asigna a un SPRITE los atributos especificados. (Ver Anexo de gráficos).

**POINT** <(x,y)>

Ofrece el código de color de un punto de la pantalla.

- <(x,y)>: coordenadas del punto. Si están fuera del rango de visualización, el valor obtenido es -1.

**BASE** <(x)>

Ofrece la primera dirección de las tablas del procesador de video (VDP)

- <(x)>: Debe ser un número entero positivo comprendido entre 0 y 19.

(Para más información, ver anexo 3 "Información Técnica").

## 11. SONIDO

- BEEP
- SOUND
- PLAY
- PLAY (N)

### BEEP

Genera el sonido «Bip». (CTRL-G).

```

10 FOR I = 1 TO 10
20 BEEP
30 FOR X = 1 TO 100: NEXT X
40 NEXT I
    
```

**SOUND** <registro del PSG>, <valor>

Escribe directamente el valor especificado en el registro indicado del Procesador generador de sonido. (PSG).

<b>Registros del PSG</b>		
<b>Nº de registro</b>	<b>Función</b>	<b>Valor</b>
0	Frecuencia canal A.	0-255
1		0-15
2	Frecuencia canal B	0-15
3		0-15
4	Frecuencia canal C	0-255
5		0-15
6	Frecuencia de ruido	0-31
7	Selecciona un canal para generación de tonos y ruido	0-63
8	Volumen canal A	0-15 La variación de volumen tendrá lugar cuando se seleccione 16
9	Volumen canal B	
10	Volumen canal C	
11	Frecuencia del patrón de variación de volumen	0-255
12		0-255
13	Selección del patrón de variación de volumen	0-14

```

10 FOR I = 1 TO 50
20 SOUND 6, 0
30 SOUND 7, 55
40 SOUND 8, 16
50 SOUND 11, 23
60 SOUND 12, 2
70 SOUND 13, 9
80 FOR X = 1 TO 100: NEXTX
90 NEXT I
100 FOR I = 1 TO 10
110 SOUND 7, 55
120 SOUND 6, 0
130 SOUND 8, 16
140 SOUND 11, 50
150 SOUND 12, 47
160 SOUND 13, 0
170 FOR X = 1 TO 100: NEXTX
180 NEXT I
190 GOTO 10

```

**PLAY** <expresión para VOZ 1>, <expresión para VOZ 2>, <expresión para VOZ 3>

Es la sentencia de ejecución de los comandos del macro lenguaje musical. Puede interpretar tres voces simultáneamente, en un rango de ocho octavas. (ver anexo II: sonido).

### **PLAY (N)**

Comprueba si se está ejecutando música.

- Si N = 1, 2, ó 3, da el valor -1 si se está ejecutando música, de lo contrario da el valor 0.
- Si N = 0, el estado de cada comando del macrolenguaje musical (-1 ó 0) se opera con OR exclusive y da el resultado obtenido.



PLAY - función que permite...  
Es la función de ejecución de las...  
Puede ejecutarse...  
de una...  
de una...

### PLAY (M)

Comanda a la...  
El valor de...  
El valor de...  
El estado de...  
El estado de...

## 12. ALMACENAMIENTO EN CASSETTE

- **CSAVE**
- **CLOAD**
- **CLOAD?**
- **SAVE**
- **LOAD**
- **BSAVE**
- **BLOAD**
- **MERGE**
- **MOTOR ON/OFF**

**CSAVE** "<Nombre del Programa>", <velocidad>

Almacena en cassette un programa **BASIC** con el nombre especificado en "<nombre del Programa>".

El formato utilizado es el Binario comprimido, que ocupa menos espacio que el ASCII, aunque algunos tipos de acceso, requieren que los ficheros estén almacenados con este formato. Por ejemplo, si queremos mezclar dos programas, uno de ellos debe estar en código ASCII. (ver comando **MERGE**).

El comando **SAVE**, realiza el almacenamiento en formato ASCII.

- "<nombre del programa>": Nombre asignado al programa en el momento de grabarlo en cassette.  
Para cargar el programa desde cas-

sette a memoria, utilizaremos este mismo nombre. El número máximo de caracteres significativos es 6.

●<velocidad>: Determina la velocidad de grabación:

1: 1.200 Baudios

2: 2.400 Baudios

Este parámetro es optativo, ya que si no se especifica, el ordenador asignará por defecto el valor 1.

CSAVE "PROG"

CSAVE "PROG1", 2

**CLOAD** "<nombre del programa>"

Carga el programa especificado en "<nombre del programa>", desde cassette a la memoria interna del ordenador, borrando el contenido actual de la memoria. Anteriormente, dicho programa fue almacenado en cinta con el comando CSAVE.

CLOAD "PROG"

**CLOAD?** "<nombre del programa>"

Carga un programa almacenado en memoria externa (cassette) a la memoria interna del ordenador sin borrar el contenido actual de la memoria interna, estableciendo una comparación entre el programa que está cargando y el que está residente en memoria.

Su función es comprobar si la grabación efectuada con CSAVE ha sido correcta. En caso de grabación incorrecta se producirá el error: "Verify error."

Una vez grabado el programa en cassette mediante CSAVE, se rebobina la cinta hasta el punto inicial de grabación y se tecldea en el ordenador **CLOAD?** seguido del nombre del programa (enmarcado entre comillas). Después, daremos la orden de ejecución pulsando la tecla **RETURN** y pondremos el cassette en marcha.

Si la grabación ha sido correcta, aparecerá «OK», en caso contrario se imprimirá el mensaje de error correspondiente.

```
CLOAD? "PROG"  
OK
```

**SAVE** "<nombre del programa>"

Almacena en cassette un programa BASIC en formato ASCII con el nombre especificado en "<nombre del programa>".

```
SAVE "PROG"
```

**LOAD** "<nombre del programa>"

Carga el programa, especificado en "<nombre del programa>", desde cassette a la memoria interna del ordenador. Anteriormente, dicho programa fué grabado con **SAVE**.

```
LOAD "PROG"
```

Si después de "<nombre del programa>" se añade: , R, el programa se ejecutará automáticamente sin necesidad de utilizar el comando **RUN**.

```
LOAD "PROG 1", R
```

**BSAVE** "<CAS: Nombre del programa>", <dirección inicial>, <dirección final>, <dirección de ejecución>.

Almacena en cassette un programa en lenguaje máquina localizado en las posiciones de memoria indicadas. (Desde <dirección inicial> hasta <dirección final>).

Si se omite el parámetro <dirección de ejecución> se asignará por defecto el valor de <dirección inicial>, sino, el programa se ejecutará a partir de esta posición cuando sea cargado con **BLOAD**.

```
BSAVE "CAS: TEST", &HA000, &HAFFF  
BSAVE "CAS: GAME", &HE000, &HE0FF, &HE020.
```

### **BLOAD** "<CAS: Nombre del programa>"

Carga desde cassette, un programa en lenguaje máquina, que fué grabado mediante **BSAVE**.

Si después de "<CAS: Nombre del programa>" se añade: , R, el programa se ejecutará inmediatamente desde la dirección que se especifica al grabarlo con **BSAVE**. El programa, se almacenará en las mismas posiciones de memoria especificados en **BSAVE**.

```
BLOAD "CAS TEST" , R
```

### **MERGE** "<nombre del programa>"

Mezcla el programa especificado en "<nombre del programa>", con el residente en memoria. El programa especificado debe ser un programa en formato ASCII (almacenado mediante **SAVE**).

Si ambos programas tienen algún número de línea común, el contenido final de esta línea será el correspondiente al programa en formato ASCII.

```
MERGE "VIDEO".
```

### **MOTOR ON/OFF**

Pone en marcha o detiene el cassette cuando se utiliza el control remoto.

Si **MOTOR** no va acompañado de argumento (**ON/OFF**) pondrá en marcha/detendrá el cassette secuencialmente. Es decir, si el cassette está parado lo pondrá en marcha y viceversa.

Si se utiliza el control remoto, todas las instrucciones de ficheros como **CLOAD**, **CSAVE**, **PRINT#**, **INPUT#**,..... etc., activarán automáticamente el cassette.

## **CONSEJOS PRACTICOS PARA LA UTILIZACION DEL CASSETTE**

- Utilice siempre un cassette monofónico. En caso de utilizar un cassette stereo, perdería parte de la información.
- Ponga el volumen del cassette a un nivel MEDIO-ALTO. La grabación estará sometida a interferencias si el volumen está demasiado bajo. Por otra parte, si el volumen está demasiado alto, puede producirse distorsión en la señal de salida.
- Si su cassette tiene control de TONO, sitúelo en una posición alta (HIGH), ya que la grabación contiene principalmente frecuencias altas.
- Compruebe que las pilas de su cassette están en buen estado. Recomendamos la utilización de un alimentador.
- Compruebe que el cabezal está en buenas condiciones de limpieza.
- Procure utilizar siempre el mismo cassette para la grabación y carga de programas, ya que entre distintos aparatos pueden existir pequeñas diferencias (velocidad del motor, ajuste de cabezales... etc.), que pueden ocasionar errores de carga.

# CONSEJOS PRÁCTICOS PARA LA UTILIZACIÓN DEL CASSETTE

- Utilice siempre un cassette reconocido. En caso de dudas en cassette usado, póngase bajo el mismo.
- Evite el volumen del cassette a un nivel alto. ALTO es el nivel con el que se debe utilizar el cassette. Si el volumen es demasiado alto, por un lado, se daña el volumen y, por otro lado, se produce deterioro en la cinta de audio.
- Si su cassette tiene un nivel de T.O.M. (Total Over Modulation) alto, ya sea la grabación o la reproducción, se producirá un ruido excesivo.
- Compruebe que las pistas de su cassette sean en buen estado. Recomendamos la utilización de un nivel alto.
- Compruebe que el cassette esté bien encajado en el aparato. Evite utilizar siempre el mismo cassette para la grabación y la reproducción. Si se utiliza el mismo cassette para la grabación y la reproducción, se daña el cassette y se produce un ruido excesivo.

### 13. FICHEROS

- OPEN #
- MAXFILES
- PRINT #
- PRINT # USING
- INPUT #
- INPUT\$(n, #)
- LINE INPUT #
- EOF
- CLOSE #

**OPEN** "<periférico>: <nombre fichero>" FOR <modo> AS # <núm. fichero>

Abre un fichero en el periférico especificado, asignándole un número determinado que será utilizado por otros comandos de Entrada/Salida para especificar el fichero con el que van a trabajar.

- <periférico>: CAS: cassette.  
CRT: pantalla.  
GRP: pantalla de gráficos.  
LPT: impresora.

- <modo>: OUTPUT: salida secuencial de datos.  
INPUT: entrada secuencial de datos.  
APPEND: adición de datos.

- <núm. fichero>: Debe ser un número entero comprendido entre 1 y el valor especificado en la sentencia MAXFILES. Este número quedará asociado al fichero en cuestión, pudiendo hacer referencia a él otros comandos de E/S (PRINT#, INPUT#,...), utilizando dicha asignación.

```
OPEN "CAS: DAT" FOR OUTPUT AS # 1
```

Abre el fichero DAT almacenado en cassette (CAS) en modo salida (OUTPUT), asignándole el número 1. (# 1)

**NOTA:** Para introducir texto trabajando con pantalla de gráfico (SCREEN 2/SCREEN 3), deberá realizar un OPEN «GRP:» FOR OUTPUT AS # 1.

```
10 OPEN "GRP:" FOR OUTPUT AS # 1
20 SCREEN 2
30 PRESET (50,100)
40 PRINT # 1, "BASIC"
50 GOTO 50
```

La Línea 30 sitúa el punto inicial de escritura.

**MAXFILES** = <núm. ficheros>

Especifica el número máximo de ficheros que estarán abiertos simultáneamente en un programa.

Si MAXFILES = 0, sólo podrán utilizarse los comandos SAVE y LOAD.

El valor asignado por defecto es 1. (MAXFILES = 1).

MAXFILES = 5, indica que en el transcurso del programa, podemos tener un máximo de 5 ficheros abiertos (OPEN) simultáneamente.

**PRINT #** <núm. fichero>, <lista de datos>

**PRINT #** <núm. fichero>, **USING** <expresión>; <lista de datos>

Escribe datos en el fichero especificado que posteriormente podrán ser leídos mediante la sentencia INPUT #.

Antes de efectuar una operación de escritura, debe abrirse el fichero en modo salida (OUTPUT).

```
10 OPEN "CAS: DAT" FOR OUTPUT AS # 1
20 INPUT # 1, PEDRO, 23
```

(Ver PRINT/PRINT USING para más detalles)

**INPUT #** <núm. fichero>, <lista de variables>

Lee datos desde el fichero especificado asignándolos a las variables indicadas en <lista de variables>.

Los datos del fichero, estarán almacenados en el mismo orden en que fueron escritos por el comando **PRINT #**, por lo que para asignarlos a las variables del programa deberemos conocer dicho orden.

- <núm. fichero>: Es el número con que fué abierto (OPEN) el fichero.
- <lista de variables>: Pueden ser numéricas o alfanuméricas, debiendo estar separadas por coma.

Antes de efectuar una operación de lectura, debe abrirse el fichero en modo entrada. (INPUT):

```
10 OPEN "CAS: DAT" FOR INPUT AS # 1
20 INPUT # 1, A$, B$, C, D$, E
```

**INPUT\$(n, #** <núm. fichero>)

Lée una secuencia de  $n$  caracteres en el fichero especificado, asignándolos a una variable.

- <número de fichero>: es el número con el que fué abierto (OPEN) el fichero.

Antes de efectuar una operación de lectura, debe abrirse el fichero en modo entrada (INPUT):

```
10 OPEN "CAS:DAT" FOR INPUT AS # 1
20 A$ = INPUT$(5, # 1)
```

**LINE INPUT #** <núm. fichero>, <variable>

Lee una línea entera (hasta 254 caracteres) del fichero especificado sin utilizar delimitadores, asignando la cadena a una variable alfanumérica.

- <núm. fichero>: es el número con el que fué abierto (OPEN) el fichero.
- <variable>: es la variable a la que se asignara la secuencia de caracteres leídos. Debe ser una variable alfanumérica.

**LINE INPUT #**, lee todos los caracteres, hasta encontrar un **RETURN**. El próximo **LINE INPUT #**, empieza a leer desde éste punto hasta el próximo **RETURN** y así sucesivamente.

Este comando es especialmente útil en aquellos programas que deben colocar cada línea de un fichero en un campo determinado o para programas que deben leer datos de otro programa almacenado en ASCII.

Antes de efectuar una operación de lectura, debe abrirse el fichero en modo entrada (**INPUT**):

```
10 OPEN "CAS: DAT" FOR INPUT AS # 1
20 LINE INPUT # 1, A$
```

**EOF** (<núm. fichero>)

EOF tomará el valor -1 si se ha llegado al final de un fichero secuencial. (EOF = -1).

Utilice EOF durante la lectura de un fichero para evitar errores tipo "Input past end".

```
IF EOF (1) = -1 THEN CLOSE # 1
```

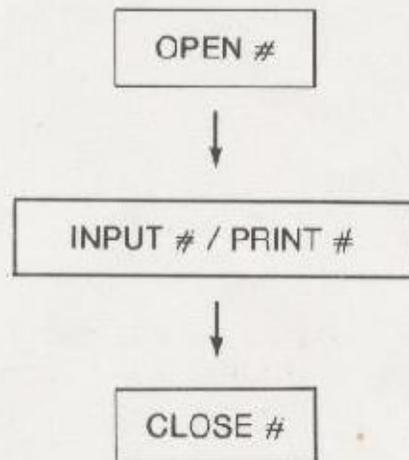
**CLOSE #** <núm. fichero>, <núm. fichero>

Cierra el fichero ó ficheros especificados en <núm. fichero>.

Si no se indica <núm. fichero>, cerrará todos los ficheros abiertos.

```
CLOSE # 1: cierra el fichero 1
CLOSE      : cierra todos los ficheros abiertos
```

**NOTA:** Las operaciones a realizar cuando trabajamos con ficheros pueden esquematizarse según el siguiente organigrama:





## **ANEXOS:**

- 1. GRAFICOS**
- 2. SONIDO**
- 3. INFORMACION TECNICA**
- 4. PROGRAMAS**
- 5. CODIGOS DE ERROR**

## ANEXOS:

1. GRAFICOS
2. SONIDO
3. INFORMACION TECNICA
4. PROGRAMAS
5. CODIGOS DE ERROR

## ANEXO 1: GRAFICOS

### INTRODUCCION

Además de las funciones gráficas vistas anteriormente (LINE, CIRCLE, PSET...), el HIT BIT nos ofrece nuevas opciones:

- Macro lenguaje para gráficos
- Gráficos SPRITE

Todas las funciones gráficas, pueden utilizarse conjuntamente, trabajando en modo gráfico (SCREEN 2, SCREEN 3).

### 1. MACRO LENGUAJE PARA GRAFICOS

Utilizando este macro lenguaje, podremos acceder a todos los puntos de la pantalla en modo gráfico (256 × 192), para realizar cualquier dibujo. Para ello, utilizaremos la sentencia DRAW, que tiene el siguiente formato:

DRAW <expresión string>

Los comandos del macro lenguaje, son letras que indican el movimiento a realizar, y forman parte de la expresión string. El string define un objeto que será dibujado al ejecutar la sentencia DRAW.

Al utilizar los comandos de movimiento (ver tabla I), hay que tener en cuenta que el movimiento se iniciará desde el último punto referenciado. Es decir, si inicialmente estamos en (5,20) y ejecutamos un R5 (desplazamiento de 5 puntos a la derecha), el punto final será (10,20), siendo éste, el punto de inicio del próximo comando.

## TABLA I: Comandos de movimiento

$n$ , indica la distancia de movimiento. El número de puntos desplazados será  $n$  veces el factor de escala. (indicando por el comando S).

$M_{x,y}$ , indica desplazamiento absoluto ó relativo. Si se indica  $X-$  ó  $X-$ , el desplazamiento será relativo, de lo contrario el desplazamiento será absoluto.

La proporción de la pantalla es 1:1; Es decir que 8 puntos horizontales tienen la misma longitud que 8 verticales.

COMANDO	MOVIMIENTO
U n	↑
D n	↓
L n	←
R n	→
E n	↘
F n	↙
G n	↗
H n	↖
$M_{x,y}$	X,Y

Todos los comandos indicados, al ser ejecutados mediante la sentencia DRAW, dibujarán siempre el recorrido programado. Si en algún momento queremos desplazarnos hasta un punto determinado sin dibujar la trayectoria seguida ó queremos dibujarla volviendo al punto inicial, utilizaremos los siguientes prefijos:

B: desplazamiento sin dibujar trayectoria

N: desplazamiento dibujando trayectoria y volviendo al punto de inicio.

Así por ejemplo, normalmente lo primero que tendremos que hacer para empezar un dibujo, será situar el cursor en la posición inicial sin dibujar la trayectoria seguida. Esto podemos hacerlo mediante el comando  $BM_{x,y}$ , donde « $x,y$ » son las coordenadas iniciales hasta donde queremos trasladarnos. ( $M_{x,y}$  dibujaría la trayectoria seguida hasta el punto  $x,y$ ).

Otros comandos que pueden utilizarse con este macro lenguaje son:

**An:** Permite movimiento en el ángulo  $n$ , que puede ser de 0 a 3 ( $0 = 0^\circ$ ,  $1 = 90^\circ$ ,  $2 = 180^\circ$ ,  $3 = 270^\circ$ ).



**Cn :** Da color a los puntos dibujados ( $n = 0 - 15$ ). Ver código de colores.

**Sn :** Indica el factor de escala. El valor de  $n$  debe ser un número entero comprendido entre 0 y 255. El factor de escala será  $n/4$ . Por ejemplo, si  $n = 1$ , el factor de escala será  $1/4$ .

El factor de escala multiplicado por la distancia indicada ( $n$ ) en los comandos de movimiento U,D,L,R,E,F,G,H,M, nos dará el número de puntos desplazados.

Si no se indica factor de escala, el valor por defecto será  $n = 4$  (S4).

**X** <variable string;>: Este comando permite ejecutar un segundo string desde otro string.

Dibujo de un cuadrado:

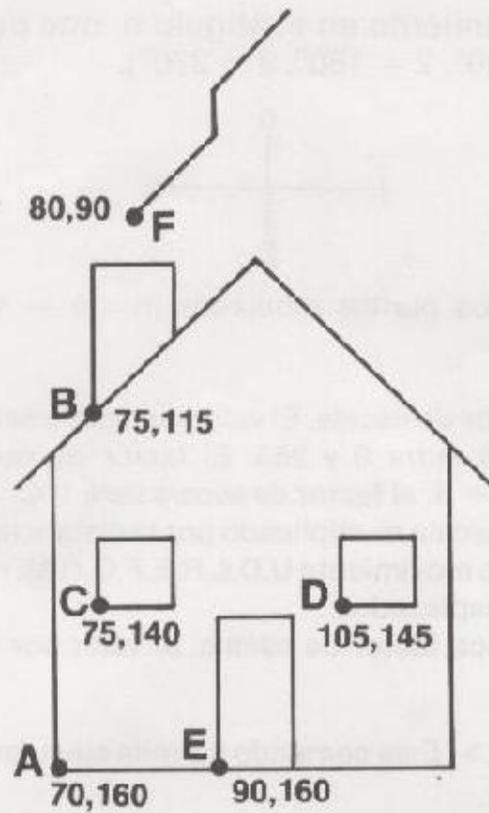
```
10 SCREEN 2
20 AS$ = "U80R80D80L80"
30 DRAW "BM70, 150XAS;"
40 GOTO 40
```

BM 70, 150 coloca el punto inicial del dibujo en (70, 150), de otra forma, el punto inicial sería (0, 0) y no podríamos visualizarlo.

El comando X, es una parte muy útil de **DRAW**, pues permite definir partes de un objeto separadas del conjunto. Así por ejemplo, si en un dibujo hay un elemento que debe aparecer en distintas posiciones podemos definir este elemento con el comando X, evitando así el tener que programarlo cada vez que queremos dibujarlo.

### Dibujo de una casa

1º) Hacemos el dibujo en una hoja de papel cuadrículada con un factor de escala de 1 cuadro = 5 puntos y fijamos el punto de inicio en (A)



2º) Hacemos el programa del dibujo, dividiéndolo para mayor comodidad en 5 partes:

A\$ = contorno casa

B\$ = chimenea

C\$ = ventana

D\$ = puerta

E\$ = humo

A\$ = "U40NGH5E25F30H5D40L50"

B\$ = "U20R10D10 "

C\$ = "U10R10D10L10"

D\$ = "U20R10D20 "

E\$ = "E10U5E10"

3º) Determinamos los puntos de inicio de cada objeto:

A\$ — CASA ————— A (70, 160)  
B\$ — CHIMENEA ————— B (75, 115)  
C\$ — VENTANA 1 ————— C (75, 140)  
C\$ — VENTANA 2 ————— D (105, 145)  
D\$ — PUERTA ————— E (90, 160)  
E\$ — HUMO ————— F (80, 90)

Antes de hacer el dibujo de cada objeto, colocaremos el cursor en los puntos correspondientes, mediante el comando **MBx,y**.

El programa completo será el siguiente:

```
10 SCREEN 2
20 A$ = 'U40NG5E25F30H5D40L50'
30 B$ = "U20R10D10"
40 C$ = "U10R10D10L10"
50 D$ = "U20R10D20"
60 E$ = "E10U5E10"
70 DRAW "BM70, 160XA$;"
80 DRAW "BM75, 115XB$;"
90 DRAW "BM75, 140XC$;"
100 DRAW "BM105, 145XC$;"
110 DRAW "BM90, 160XD$;"
120 DRAW "BM80, 90XE$;"
130 GOTO 130
```

Para mayor comodidad podemos hacer todo el dibujo con una sola instrucción. Para ello sustituiríamos las líneas 70-130 por:

```
70 DRAW "BM70, 160XA$; MB 75, 115 XB$; BM 75, 140XC$; BM 105,
145 X C$; BM 90, 160 XD$; BM 80, 90 XE$;"
80 GOTO 80
```

PROGRAMA EJEMPLO UTILIZACION MACRO LENGUAJE PARA  
GRAFICOS

5 REM \$MACRO LENGUAJE GRAFICOS\$

10 SCREEN2

20 REM:A\$ = PERFIL CASA.

30 A\$ = "BM70,130U60BD60R60U60BL30BU40NF40NG40D2NF38NG38"

40 REM:B\$ = PUERTAS/VENTANAS.

50 B\$ = "BM80,70R10D20L10U20BR30R10D20L10U20BM90,  
100R20D30BL20U30BM82,70D20BR2U20BR2D20BR2U20BM112,  
70D20BR2U20BR2D20BR2U20BM92,100D30BR2U30BR2D30BR2U30BR2  
D30BR2U30BR2D30BR2U30BR2D30"

55 REM:C\$ = CHIMENEA/HUMO.

58 C\$ = "BM8050U15R10D5BM90,30E10U10E10BD10R10D10  
L10U10DB10BR20U10F10U10BR10F5NE5D5"

60 REM:D\$ = PAJAROS.

70 D\$ = "BM170,30F10E10BM190,40F10E10"

80 REM:E\$ = CARTEL HIT BIT 55.

90 E\$ = "BM170,120R60D50L60U50BM180,  
140 U10D5R5D5BR5U5BU2U1BM195,  
140 U10NL2NR2BR10D10U5R5D5L5BR10U5BU2U1BM220,  
140 U10NL2NR2BM190,160R5U5L5U5R5BD10BR10R5U5L5U5R5BM190,  
130 D20R20U20"

100 REM:F\$ = CAMINO.

110 F\$ = "BM90,140F10G10F10BM110,140F10G10F10"

150 DRAW"XA\$;"

160 DRAW"XB\$;"

170 DRAW"XC\$;"

```

180 DRAW "XD$;"
190 DRAW "XE$;"
200 DRAW "XF$;"
250 FOR I = 0 TO 5000
260 NEXT I
270 A = RND(1)*15
280 B = RND(1)*15:C = RND(1)*15
290 COLOR A,B,C
300 GOTO 10

```

## 2. GRAFICOS SPRITE

Hasta ahora, para representar gráficos en movimiento hemos utilizado la sentencia **PRINT**. Esta forma de representación para la animación gráfica, resulta muy limitada, debido principalmente a que al utilizar la sentencia **PRINT**, los objetos representados deben crearse a partir de los símbolos gráficos existentes, por lo que la forma y resolución del objeto, no puede ser lo buena que se requiere.

Por otra parte, el movimiento del gráfico por la pantalla requiere un número de instrucciones para controlar su situación.

Veamos un ejemplo: el siguiente programa simula el vuelo de una nave espacial:

```

10 CLS
20 FOR A = 0 TO 26
30 LOCATE A,10
40 PRINT "XOX"
50 FOR I = 1 TO 40
60 NEXT I
70 LOCATE A, 10
80 PRINT " "
90 NEXT A

```

Al ejecutar el programa en el ordenador, se observa que la nave espacial (XOX) no ha salido muy favorecida, debido a las causas citadas anteriormente, y el control de su movimiento puede llegar a complicar bastante el programa si pretendemos que el objeto se mueva por distintas direcciones... Todos estos inconvenientes quedan subsanados utilizando los gráficos **SPRITE** del **HIT BIT**, ya que podemos dar al objeto la forma deseada, situarlo en cualquier posición de la pantalla, darle una dirección y velocidad,... etc.

### ¿Qué es un SPRITE?

Un SPRITE, es un objeto programable, que puede definirse fácilmente en **BASIC**.

SPRITE, es el nombre que recibe la forma de lograr imágenes más naturales, y más fácilmente en un ordenador. En los gráficos convencionales, las imágenes se componen en una pantalla simple, como si se dibujara en una hoja de papel. Con los SPRITES, el ordenador tiene varios planos, cada uno de los cuales contiene su propia imagen. En el caso del **HIT BIT**, disponemos de 32 planos distintos.

La manera más sencilla de representar estos planos, consiste en imaginarse láminas de plástico transparente. Si la lámina más cercana al observador contiene el dibujo de una estrella, mientras que la última incluye la imagen de una nave espacial, ésta será vista pasar por detrás de la estrella, como si realmente estuviera flotando en el espacio.

Al poner varios elementos en planos separados, podrán crearse efectos tridimensionales muy convincentes.

Otra característica importante de este tipo de gráficos es que para darles movimiento tan solo hay que especificar una velocidad y una dirección, olvidándonos por completo de cómo fue definido.

Veamos como mejora el aspecto de nuestra nave espacial utilizando SPRITES; Ejecute el siguiente programa:

```
10 COLOR 15, 4, 7
20 SCREEN 2,0
30 CLS
40 SPRITE $(0) = CHR$(&B00000000) +
+ CHR$(&B00000000) + CHR$(&B00100100) +
+ CHR$(&B00111100) + CHR$(&B01011010) +
```

```

+ CHR$(&B10011001) + CHR$(&B01111110) +
+ CHR$(&B00100100)
50 PUT SPRITE 0, (5, 70)
60 FOR I = 1 TO 256
70 PUT SPRITE 0, STEP (1,0), 15
80 NEXT I

```

### Comandos para la utilización de SPRITES en el HIT BIT

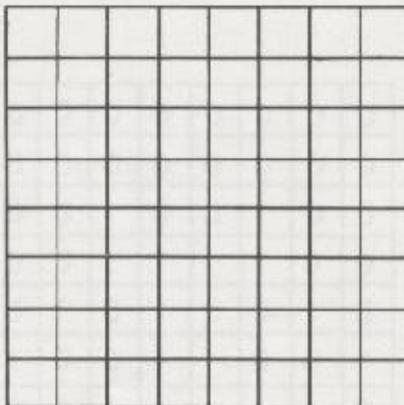
Los comandos utilizados en el **HIT BIT** para la realización de gráficos SPRITE, son los siguientes:

- SPRITE \$(n)
- PUT SPRITE
- ON SPRITE GOSUB
- SPRITE ON/OFF/STOP

**SPRITE \$(n) = <definición SPRITE>**

Define la forma del objeto. El número de plano (n) debe ser un número entero entre 0 y 31.

Para definir el SPRITE, nos auxiliaremos de una cuadrícula de 8 × 8:



Dentro de esta cuadrícula, dibujaremos el **SPRITE** y a continuación, al lado de cada línea, codificaremos su contenido según el siguiente criterio:

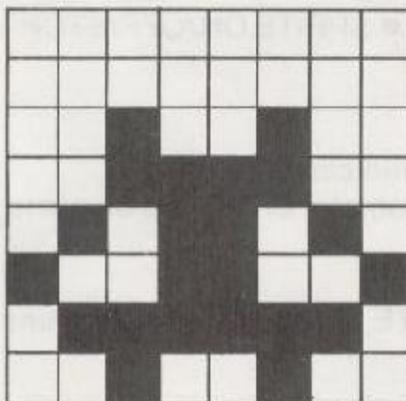
- Si una casilla está en blanco, le asignaremos el valor "0".
- Si una casilla está pintada, le asignaremos el valor "1".

Finalmente, como estos valores obtenidos son binarios, se lo indicaremos al ordenador mediante el prefijo "&B" (si conoce el lenguaje binario, puede codificar estos valores en decimal). La definición del **SPRITE**, estará formada por la suma de los 8 CHR\$ correspondientes a los valores decodificados.

Veamos como construiríamos la nave espacial del ejemplo anterior:

- 1º Hacemos el dibujo en la cuadrícula.
- 2º Decodificamos las líneas de la cuadrícula.
- 3º Definimos el **SPRITE** como la suma de los 8 CHR\$ decodificados.

### 1º Dibujo



### 2º Decodificación

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	1	0	1	1	0	1	0
1	0	0	1	1	0	0	1
0	1	1	1	1	1	1	0
0	0	1	0	0	1	0	0

### 3º Asignación

```
SPRITE$(0) = CHR$( &B00000000) -  
+ CHR$( &B00000000) + CHR$( &B00100100) +  
+ CHR$( &B00111100) + CHR$( &B01011010) +  
+ CHR$( &B10011001) + CHR$( &B01111110) +  
+ CHR$( &B00100100)
```

Una forma más cómoda de definir el SPRITE es utilizando *READ/DATA*:

```
10 COLOR 15,4,7  
20 SCREEN 2,0  
30 CLS  
40 FOR X = 1 TO 8  
50 READ A$(X)  
60 B$ = B$ + CHR$( VAL("&B' + A$(X))  
70 NEXT X  
80 SPRITE$(0) = B$  
90 PUT SPRITE 0, (5,70)  
100 FOR I = 1 TO 256  
110 PUT SPRITE 0, STEP (1,0), 15  
120 NEXT I  
130 DATA 00000000  
140 DATA 00000000  
150 DATA 00100100  
160 DATA 00111100  
170 DATA 01011010  
180 DATA 10011001  
190 DATA 01111110  
200 DATA 00100100
```

Si queremos que nuestra «nave espacial» aparezca más grande, sustituiremos la línea 20 por:

```
20 SCREEN. 2,1 (ver sentencia SCREEN)
```

También podemos trabajar con **SPRITES** de  $16 \times 16$  (ampliables con **SCREEN 2,3**). El procedimiento será el mismo que en  $8 \times 8$ , teniendo en cuenta que la cuadrícula será de  $16 \times 16$  y por lo tanto tendremos 16 líneas de codificación de 16 bits cada una. El principio del programa, debemos indicar si trabajamos con **SPRITES** de  $8 \times 8$  ó  $16 \times 16$  y si van a ser ampliables:

```
SCREEN X, 0 — 8 × 8 normal  
SCREEN X, 1 — 8 × 8 ampliado  
SCREEN X, 2 — 16 × 16 normal  
SCREEN X, 3 — 16 × 16 ampliado
```

#### **PUT SPRITE**

Con esta sentencia definimos los atributos del **SPRITE** (situación, movimiento, color...). Su estructura es la siguiente:

```
PUT SPRITE N, (X, Y), <Color>, <n° SPRITE>
```

N = n° plano (0 - 31)

(X, Y) = situación del **SPRITE**

Color = color del **SPRITE**

n° **SPRITE** = indica el **SPRITE** al que vamos a asignar los atributos anteriores. Este número debe ser menor a 256 si el tamaño del **SPRITE** es 0 ó 1 y menor a 64 si el tamaño es 2 ó 3 (ver sentencia **SCREEN**)

Para poner el **SPRITE** en movimiento, le indicaremos los saltos (**STEPS**) que debe realizar a partir de su posición inicial. Así pues, normalmente utilizaremos 2 veces la sentencia **PUT SPRITE**: una para colocarlo en la posición inicial y otra para indicarle el movimiento a realizar.

En el ejemplo II, la línea 50 coloca el **SPRITE** en la posición inicial (5,70) y en la línea 70 le especificamos que desde la posición anterior, debe moverse con saltos de (1,0). (desplazamiento horizontal). Si en lugar de **STEP (1,0)** especificamos **STEP (0,1)**, el movimiento será vertical.

Si queremos que nuestra nave espacial se mueva siguiendo la trayectoria de una senoide, cambiaremos las siguientes líneas de programa del ejemplo anterior:

```
60 FOR X = 11 TO 256 STEP 3
70 PUT SPRITE 0, (X, SIN (X/4)*20 + 70)
```

Con las siguientes modificaciones obtendremos un movimiento aleatorio:

```
50 PUT SPRITE 0, (127,96)
60 X = INT (RND(1)*20)*(-1)^ Y:Y = INT(RND(1)*20)*(-1)^X
70 PUT SPRITE 0, STEP (X,Y)
80 FOR T = 1 TO 50: NEXT
90 GOTO 60
```

#### **ON SPRITE GOSUB <nº línea>**

Esta sentencia hará que el programa principal salte a la subrutina de servicio, cuya línea de inicio es la indicada en <nº de línea>, cuando «chocan» dos **SPRITES**, si anteriormente se ha habilitado la interrupción mediante **SPRITE ON**.

#### **SPRITE ON/OFF/STOP**

Para activar/desactivar las interrupciones de **SPRITES**.

**SPRITE ON:** Activa la interrupción

**SPRITE OFF:** Desactiva la interrupción

**SPRITE STOP:** Si se produce interrupción, la memoriza, pero no salta a la subrutina de servicio hasta que se active la interrupción mediante **SPRITE ON**

Antes de la sentencia **ON SPRITE GOSUB**, hay que especificar si la interrupción está activa o no. (**SPRITE ON/OFF/STOP**). Si se produce interrupción, con **SPRITE ON**, automáticamente se producirá un **SPRITE STOP**, y pasaremos a realizar la subrutina de servicio.

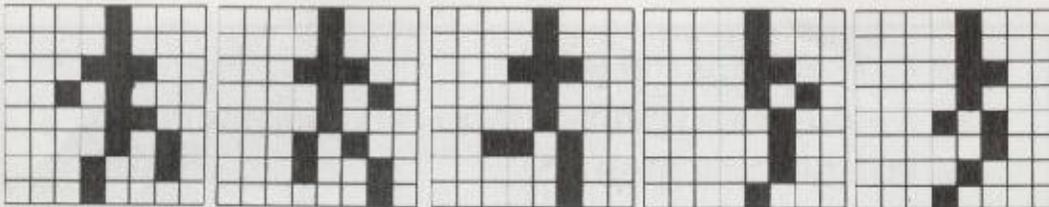
#### **VARIACION DE LA FORMA DEL OBJETO CON EL MOVIMIENTO**

Hasta aquí, hemos visto cómo mover un **SPRITE** por la pantalla sin variar su forma inicial. Si queremos que nuestro objeto dé más sensación de

movimiento (moviendo las piernas y los brazos al andar, por ejemplo), tendremos que crear tantos **SPRITES** como movimientos distintos tengamos. Es decir, tendremos que hacer un estudio del movimiento a realizar y definir un **SPRITE** para cada movimiento (imagínese el movimiento en cámara lenta).

Veamos como visualizar un **SPRITE** que está «corriendo»:

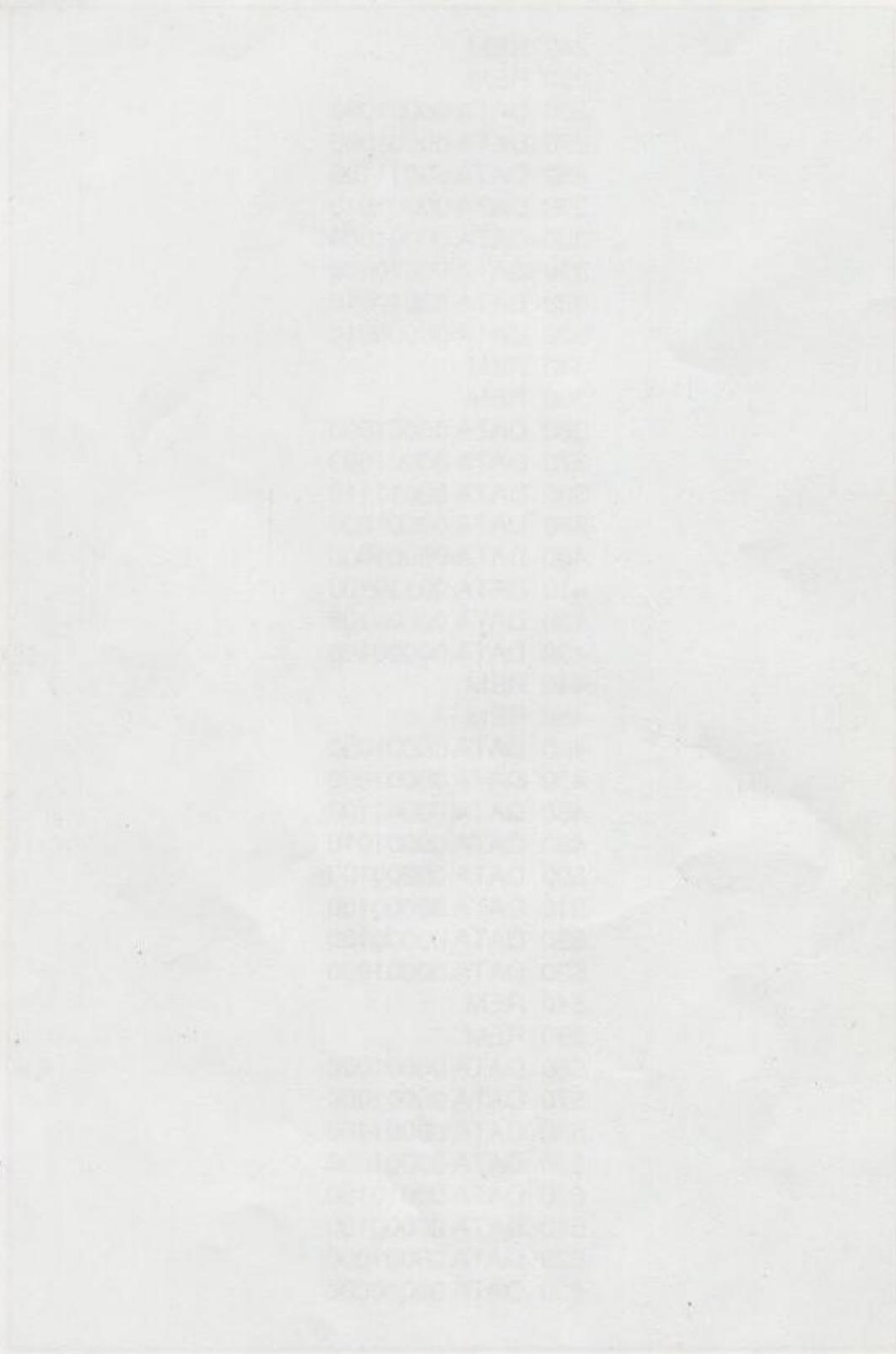
Los movimientos realizados al correr serán 5:



Por lo tanto, utilizaremos 5 SPRITES (SPRITES (0) - SPRITES (4)).  
El programa será el siguiente:

```
10 COLOR 15,4,7
20 SCREEN 1,0
30 CLS
40 FOR L=0 TO 4
50 FOR I=1 TO 8
60 READ D$(L)
70 S$(L) = S$(L)+CHR$(VAL("&B"+D$(L)))
80 NEXT I
90 SPRITE$(L)=S$(L)
100 NEXT L
100 PUT SPRITE 0,(120,90),15,0
120 FOR I = 0 TO L-1
130 PUT SPRITE 0, STEP(2,0),15,I
140 FOR T = 1 TO 20:NEXT
150 NEXT I:GOTO 120
160 DATA 00001000
170 DATA 00001000
180 DATA 00011100
190 DATA 00101000
200 DATA 00001100
210 DATA 00001010
220 DATA 00010010
230 DATA 00010000
```

240 REM  
250 REM  
260 DATA 00001000  
270 DATA 00001000  
280 DATA 00011100  
290 DATA 00001010  
300 DATA 00001000  
310 DATA 00010100  
320 DATA 00010010  
330 DATA 00000010  
340 REM  
350 REM  
360 DATA 00001000  
370 DATA 00001000  
380 DATA 00011110  
390 DATA 00001000  
400 DATA 00001000  
410 DATA 00110100  
420 DATA 00000100  
430 DATA 00000100  
440 REM  
450 REM  
460 DATA 00001000  
470 DATA 00001000  
480 DATA 00001100  
490 DATA 00001010  
500 DATA 00000100  
510 DATA 00000100  
520 DATA 00000100  
530 DATA 00001000  
540 REM  
550 REM  
560 DATA 00001000  
570 DATA 00001000  
580 DATA 00001100  
590 DATA 00001000  
600 DATA 00010100  
610 DATA 00000100  
620 DATA 00001000  
630 DATA 00010000



## ANEXO 2: SONIDO

### INTRODUCCION

Una de las características principales del HIT BIT es que internamente, lleva incorporado un Procesador generador de sonido que le permitirá crear fácilmente sus propias melodías utilizando un Macro lenguaje Musical, muy similar al visto anteriormente para dibujar. Para ello dispondremos de 8 octavas, 3 Tonos (3 voces simultáneas) y una señal de ruido (para crear efectos especiales).

### MACRO LENGUAJE MUSICAL

La sentencia de ejecución es PLAY, que realiza una función similar a la desempeñada por DRAW, utilizando un macro lenguaje musical dentro de una expresión STRING. El formato es el siguiente:

```
PLAY <exp. String para VOZ 1> , <exp. String para VOZ 2> , <exp. String para VOZ 3>
```

<exp. String para VOZ N> , es una expresión String consistente en comandos musicales de una sola letra. Si no se especifica alguna de las 3 voces, el canal correspondiente permanecerá en silencio.

Las notas musicales pueden especificarse de dos formas distintas:

1º) Utilizando la notación alfabética convencional

Las notas se representan mediante las letras A-G:

C	D	E	F	G	A	B
do	re	mi	fa	sol	la	si

Para indicar un sostenido, colocaremos el símbolo # ó + a continuación de la nota. Por ejemplo Do sostenido se representará como C # ó C+.

Para indicar un bemol, representaremos la nota con el símbolo —. Así D— será Re bemol.

Únicamente pueden utilizarse los sostenidos y bemoles que corresponden a las teclas negras del piano, no siendo válidas las siguientes expresiones:

● B #	(Si sostenido)	=	C (Do)
● C—	(Do bemol)	=	B (Si)
● E #	(Mi sostenido)	=	F (Fa)
● F—	(Fa bemol)	=	E (Mi)

Para indicar la octava correspondiente, utilizaremos el comando **On** (n = 1-8).

Cada octava va de Do a Si (C —B). Si no se especifica octava, se entenderá 04.

PLAY "C D E F G A B" ← Escala natural

PLAY "CC # DD # EFF # GG # AA # B" ← Escala cromática

```

10 REM$ ACOMPANAMIENTO ROCK $
20 A$ = "T23003EG#BO4C#DC#O3BG#"
30 B$ = "T23003AO4C#EF#GF#EC#"
40 C$ = "T23003BO4D#F#G#AG#F#D#"
50 FOR I = 1 TO 2 : PLAY A$ : NEXT
60 PLAY B$ : PLAY A$
70 PLAY C$ : PLAY B$
80 GOTO 50
    
```

T 230, indica la velocidad de ejecución (ver comando T)

2º) Utilizando una notación numérica (Nn)

En lugar de designar las notas con letras (A-G), lo hacemos mediante números: Nn (n = 0 - 96). Esta es una forma de seleccionar las notas sin tener que especificar nombre (A-G), octava ni alteraciones (sostenidos y bemoles). La octava 4 corresponderá a la siguiente numeración:

NOTA	NOTACION NUMERICA	NOTACION ALFABETICA
DO	36	C
DO #	37	C #
RE	38	D
RE #	39	D #
MI	40	E
FA	41	F
FA #	42	F #
SOL	43	G
SOL #	45	G #
LA	45	A
LA #	46	A #
SI	47	B

Cada octava está formada por 12 notas por lo que para calcular el número correspondiente a una nota en otra octava, tendremos que sumar/restar 12.

Así por ejemplo, DO (Octava 4) = 36 (ver Tabla):

DO (Octava 5) = 36 + 12 = 48

DO (Octava 3) = 36 - 12 = 24

10 REM \$ NOTACION NUMERICA \$

20 A \$ = "N36N37N38N39N40N41N42N43N44N45N46N47"

30 PLAY A \$

### COMANDOS MUSICALES

Dentro de las expresiones anteriores, pueden utilizarse también los siguientes comandos:

**Ln** : Duración de las notas (n = 1 - 64)

La duración será 1/n

Ln	Duración
L1	1
L2	1/2
L3	1/3
L4	1/4
"	"
"	"
L64	1/64

Si dentro de una expresión queremos variar la duración de una nota, lo haremos colocando el valor al lado de ésta. Así por ejemplo: A16 equivale a L16A, pero L16AB es distinto de A16B (debería ser A16B16).

**Rn** **Pausa** ( $n = 1-64$ ). Se utiliza para simular los "silencios". La duración se calcula del mismo modo que para Ln.

**Punto.** Realiza la operación del "puntillo" (la duración de la nota precedente se multiplica por  $3/2$ ). Puede aplicarse también a la pausa (Rn).

**Ejemplo:** G2. = GGG (duración de G =  $2 \times 3/2 = 6/2 = 3$ ).

**Tn** **Tiempo:** velocidad de ejecución ( $n = 32 - 255$ )  
Si no se indica, se entenderá T12 $\emptyset$ .

**Vn** **Volúmen:** volúmen de salida ( $n = \emptyset - 15$ )

**Mn** **Modulación:** período de la envolvente ( $n = \emptyset - 65.535$ ).

**Sn** **Forma:** forma de la envolvente ( $n = \emptyset - 15$ ):

r	Forma de la envolvente
0, 1, 2, 3, 9	
4, 5, 6, 7, 15	
8	
13	
11	
12	
13	
14	

**X <variable>**: Ejecuta el String especificado.

En todos estos comandos, el argumento n puede ser constante, como por ejemplo 17, ó variable. En tal caso, lo escribiremos de la forma: " = < variable >; ", donde variable es el nombre de una variable (X por ejemplo).

**Ejemplo:** Este programa, nos pedirá que le entremos el valor del período de la envolvente y nos ejecutará una pequeña melodía con las distintas formas de envolvente (ver Tabla II).

```
10 INPUT "PERIODO:"; B
20 AS = T23003EG#B04C#DC#03BG#"
30 FOR A = 3 TO 15
40 PLAY "M=B;" : PLAY "S= A;"
50 PLAY A$
60 NEXT
```

X <variable> : Ejecuta el string especificado

En todos estos comandos el argumento a utilizar es el nombre de la variable. En tal caso se ejecutará el comando en la forma: " = <variable> ", donde variable es el nombre de la variable (X por ejemplo).

Ejemplo: Este programa nos permite ver la evolución de la población de la provincia de la provincia de la provincia y nos muestra una pantalla similar con las distintas formas de crecimiento (ver Tabla 1).

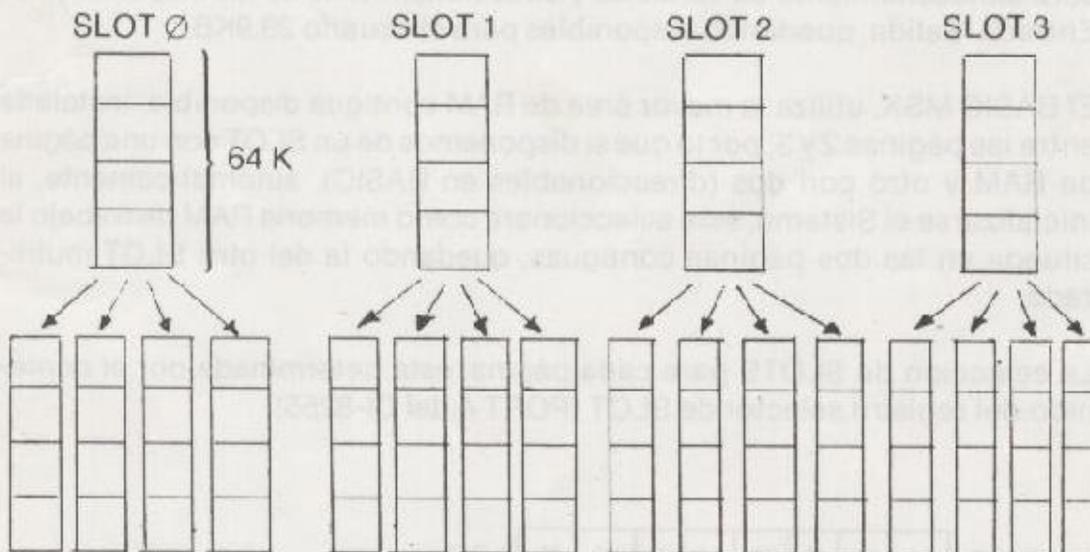
```
10 INPUT "PERIODO: "; B
20 AS = TZMUSE + B * 1000000
30 FOR A = 1 TO 12
40 PLAY "M-B"; BAY "S-A"
50 PLAY AS
60 NEXT
```

## ANEXO 3: INFORMACION TECNICA

### 1. MAPA DE MEMORIA

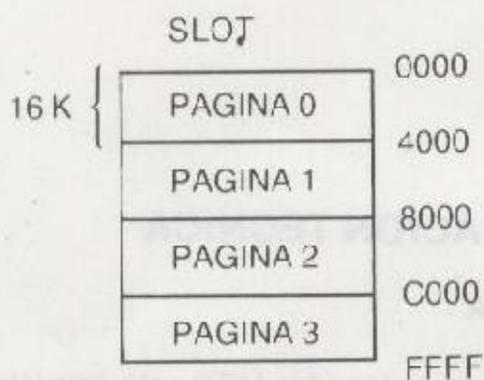
El Mapa de memoria de un ordenador MSX está estructurado en cuatro bloques de 64K, denominados SLOTS "primarios", con lo que se dispone de un espacio físico en memoria de 256K.

Cada uno de estos SLOTS puede ampliarse hasta un máximo de cuatro (SLOTS secundarios), disponiendo así de un espacio en memoria de 1 Megabyte. (espacio físico total).



El microprocesador Z80, tiene 16 bits de direccionamiento (BUS de direcciones), por lo que directamente sólo puede acceder a 64K. ( $2^{16} = 65.536 = 64K$ ) Para permitir el acceso a todo el espacio de memoria disponible, cada SLOT se divide en cuatro páginas lógicas de 16K, y mediante el registro selector de SLOT (PORT A del CI-8255), se transforma el espacio físico de

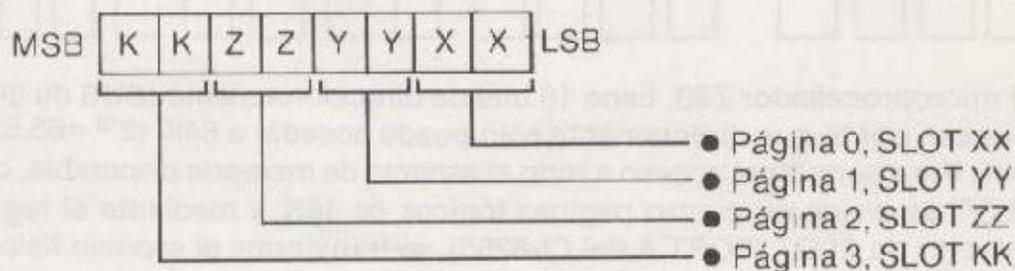
memoria en un bloque lógico de 64K formado por páginas de 16K de los distintos SLOTS. De esta forma, la memoria física está siempre situada en el espacio de direccionamiento de la CPU.



La ROM del BASIC MSX (BIOS + Interpretador BASIC MSX), ocupa las páginas 0 y 1 del SLOT 0, por lo que la máxima memoria RAM direccionable, trabajando en BASIC, es de 32KBytes. De estas 32KB, el sistema se reserva 4K para almacenamiento de variables y direccionamiento de los dispositivos de Entrada/Salida, quedando disponibles para el usuario 28,9KB.

El BASIC MSX, utiliza la mayor área de RAM contigua disponible, instalada entre las páginas 2 y 3, por lo que si disponemos de un SLOT con una página de RAM y otro con dos (direccionables en BASIC), automáticamente, al inicializarse el Sistema, éste seleccionará como memoria RAM de trabajo la situada en las dos páginas contiguas, quedando la del otro SLOT inutilizada.

La selección de SLOTS para cada página, está determinada por el contenido del registro selector de SLOT (PORT A del CI-8255).



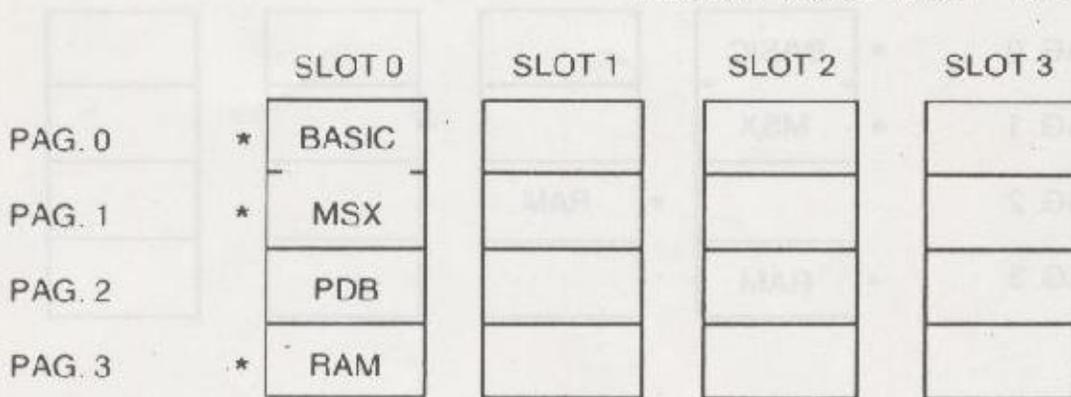
Con el valor de este registro, podemos hacer un esquema del Mapa de memoria en cada configuración:

1º) HIT BIT 55

● Registro selector de SLOT:

00	00	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

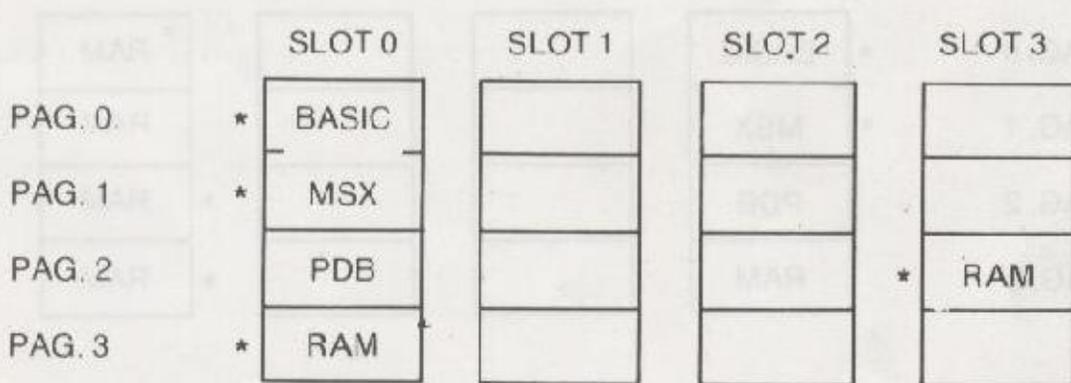


2º) HIT BIT 55 + HBM 16 (conector posterior HIT BIT)

● Registro selector de SLOT:

00	11	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

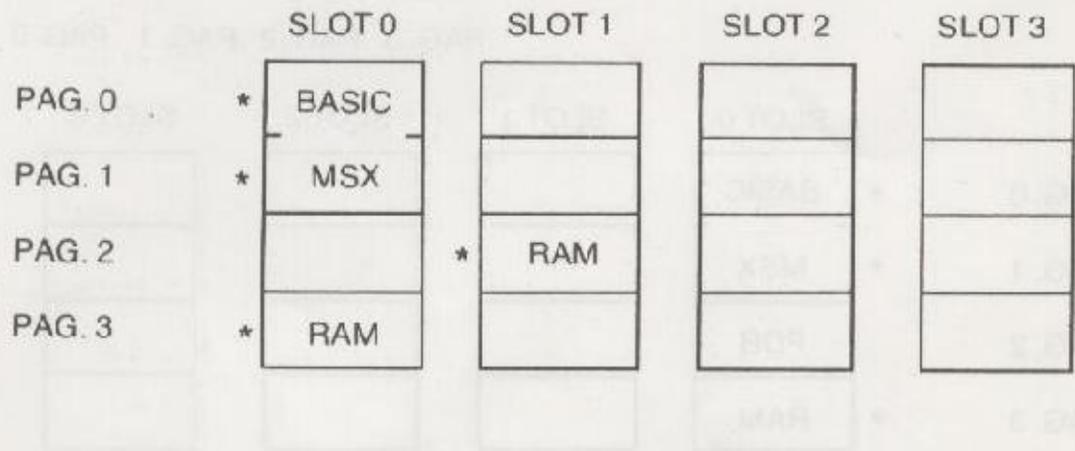


3°) HIT BIT 55 + HBM 16 (conector superior)

- Registro selector de SLOT:

00	01	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

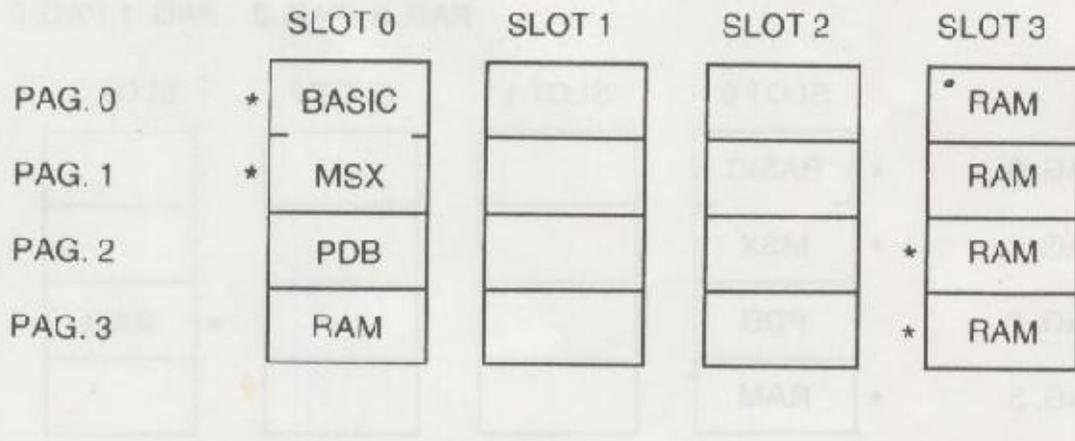


4°) HIT BIT 55 + HBM 64 (conector posterior)

- Registro selector de SLOT:

11	11	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

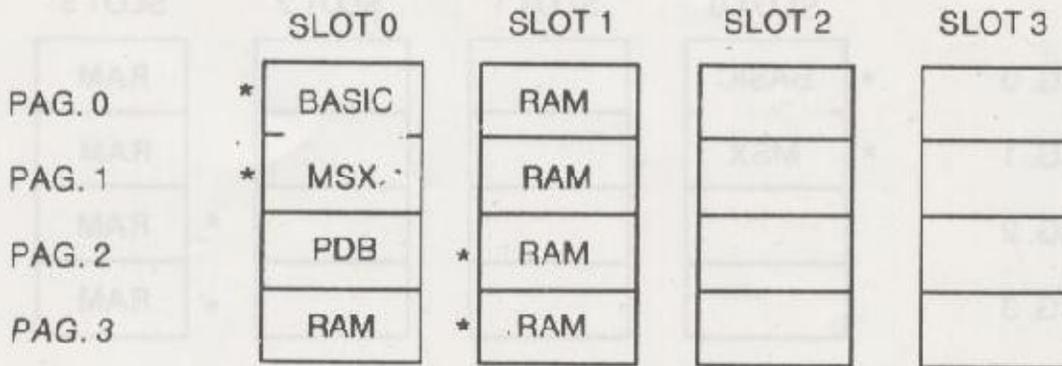


5º) HIT BIT 55 + HBM 64 (conector superior)

● Registro selector de SLOT:

01	01	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

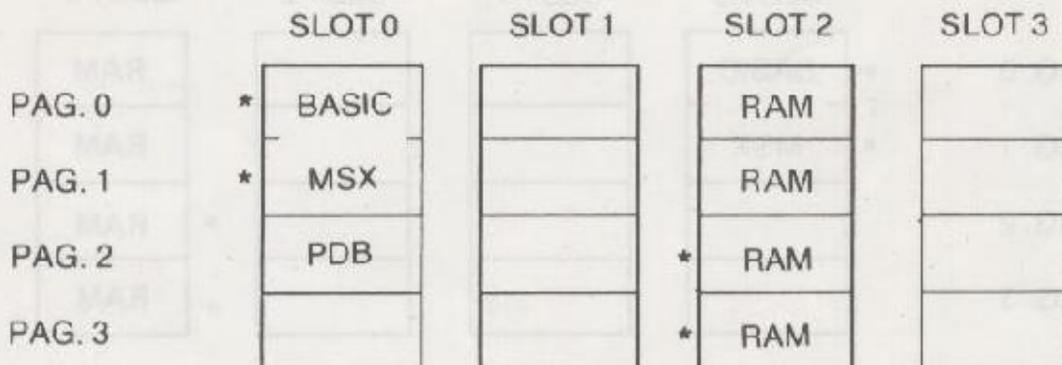


6º) HIT BIT 75

● Registro selector de SLOT:

10	10	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

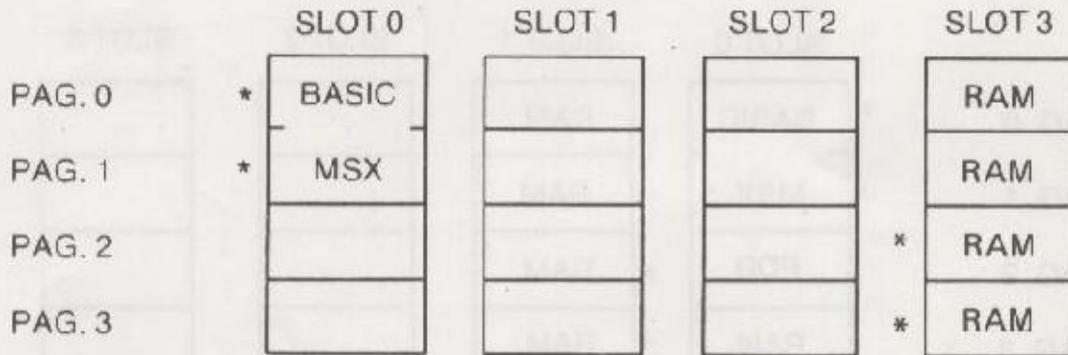


## 7º) HIT BIT 10P

- Registro selector de SLOT:

11	11	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

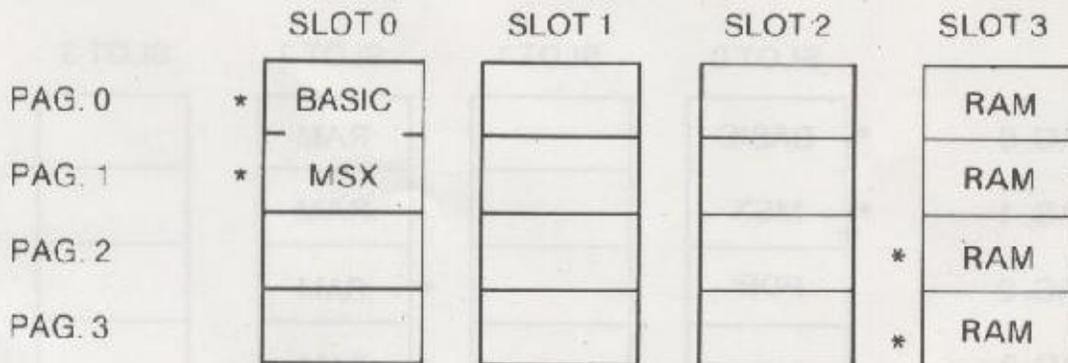


## 8º) HIT BIT 20P

- Registro selector de SLOT:

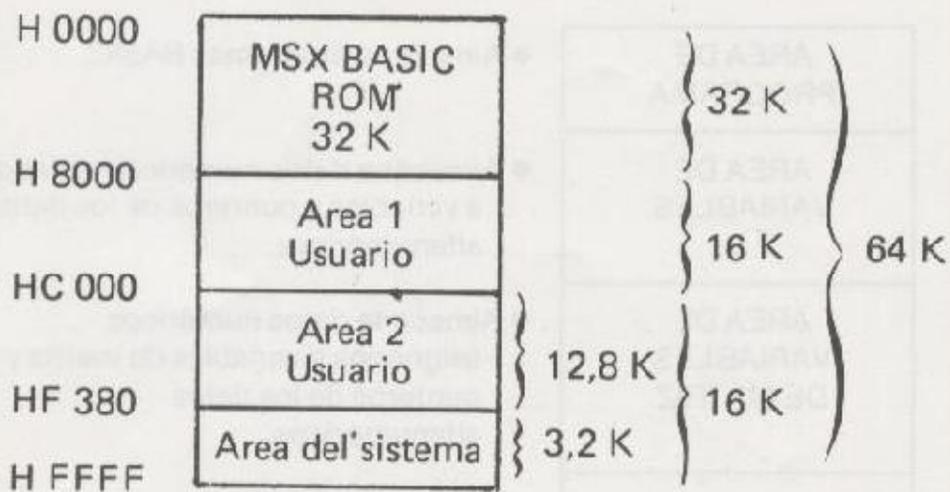
11	11	00	00
----	----	----	----

PAG. 3 PAG. 2 PAG. 1 PAG. 0

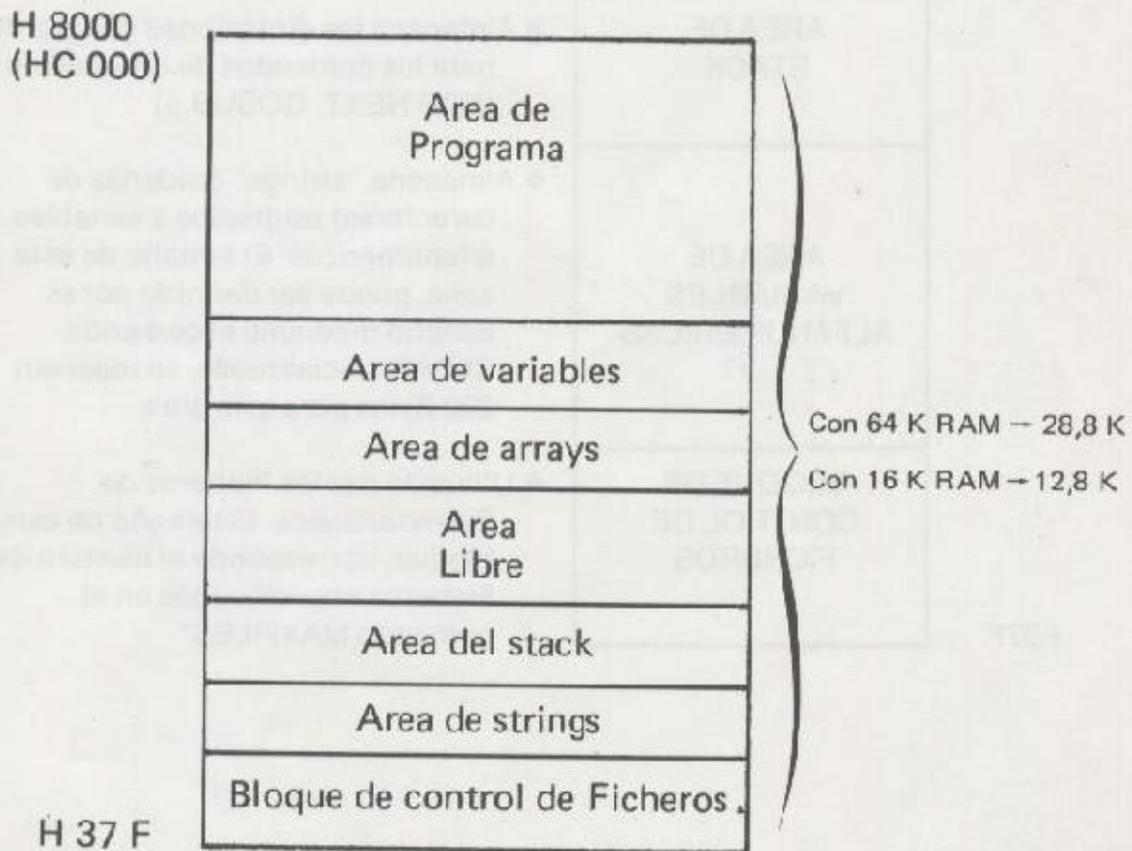


# MAPA DE MEMORIA

## I. Disposición general trabajando en BÁSIC-MSX:



## II. Area disponible por el usuario:



## CONFIGURACION DEL AREA DE USUARIO

Trabajando en BASIC, la RAM de usuario tiene la siguiente estructura:

8000 (C000)	AREA DE PROGRAMA	● Almacena programas BASIC.
	AREA DE VARIABLES	● Almacena datos numéricos aislados a variables y punteros de los datos alfanuméricos.
	AREA DE VARIABLES DE MATRIZ	● Almacena datos numéricos asignados a variables de matriz y punteros de los datos alfanuméricos.
	AREA LIBRE	● Zona de memoria libre disponible. El tamaño de esta zona puede conocerse utilizando la función FRE.
	AREA DE STACK	● Almacena las direcciones de retorno para los comandos de bifurcación. (FOR NEXT, GOSUB...)
	AREA DE VARIABLES ALFANUMERICAS	● Almacena "strings" (cadenas de caracteres) asignados a variables alfanuméricas. El tamaño de esta zona, puede ser definido por el usuario mediante el comando CLEAR. Inicialmente, se reservan 200 Bytes para esta área.
F37F	BLOQUE DE CONTROL DE FICHEROS	● Utilizado por los Ficheros de Entrada/Salida. El tamaño de este bloque, corresponde al número de Ficheros especificados en el comando MAXFILES.

## OBSERVACIONES

- Normalmente, los SLOTS 0 y 2 son internos al sistema:
  - SLOT 0, Página 0 y Página 1: BASIC MSX
  - SLOT 0, Página 2 : PERSONAL DATA BANK
- La dirección de memoria FFFF de los SLOTS «primarios» contiene la situación del Registro de selección de SLOT «secundario». El valor real de este Registro es el complemento de su valor actual.
- El símbolo "\*", indica las páginas direccionadas en BASIC, que corresponden a las indicadas en el Registro selector de SLOT.
- Todos los ordenadores MSX disponen de 16K de memoria RAM de Video (VRAM), que se utiliza exclusivamente para la visualización en pantalla, por lo que no se incluyen en el Mapa de Memoria.
- EL Sistema Operativo de Disco, MSX.DOS requiere 64K de memoria de RAM.

MAPA DE ENTRADA/SALIDA

DIRECCION

FF	Area reservada	Dirección	L/E	Aplicación	OBSERVACIONES	
E0		80	L	Lectura Datos	INTERFACE RS 232C	
			E	Ent. Datos		
D8	Controlador Lector diskettes	81	L	Estado		
			E	Est. Modo		
D0	Area Reservada	90	L	Lectura "Status" (B0)		IMPRESORA
			E	Salida "Strobe" (B1)		
B0	Interface Programable (PPI)	91	E	Ent. Datos		
			98	L		Lectura datos en Ram de video
E	Ent. datos en Ram de video					
A8	Procesador de sonido	99	L	Estado		
			E	Establecimiento Mandatos/Direcc.		
A0	Procesador de video	A0	E	Mantenimiento de Dirección	PROCESADOR DE SONIDO AY-3-8910	
			A1	E		Ent. datos
			A2	L		Lectura datos
98	Impresora	A8	L	Lectura datos Port A	PP 8255	
			E	Ent. datos Port A		
90	Area Reservada	A9	L	Lectura datos Port B		
			E	Ent. datos Port B		
88	RS 232C	AA	L	Lectura datos Port C		
			E	Ent. datos Port C		
80	Area Reservada	AB	E	Est. Modo trabajo		
			00			

L = LECTURA  
E = ESCRITURA

DISTRIBUCION DE LOS PORTS DE E/S DEL PPI (8255)

PORT	BIT	E/S	SEÑAL	APLICACION
A	0	S	CS0L	Selección de Slot en Página 0
	1	S	CS0H	
	2	S	CS1L	Selección de Slot en Página 1
	3	S	CS1H	
	4	S	CS2L	Selección de Slot en Página 2
	5	S	CS2H	
	6	S	CS3L	Selección de Slot en Página 3
	7	S	CS3H	
B	0	E		Teclado
	1	.		
	2	.		
	7	E		
C	0	S	KB0	Teclado
	1	S	KB1	
	2	S	KB2	
	3	S	KB3	
	4	S	CASON	Control de cassette (0=ON)
	5	S	CASW	Escritura cassette
	6	S	CAPS	Lámpara Caps (0=ON)
	7	S	SOUND	Salida de sonido (cuando está controlado por Soft)

Página 0: 0000 - 3FFF

Página 1: 4000 - 7FFF

Página 2: 8000 - BFFF

Página 3: C000 - FFFF

E = Entrada

S = Salida

DISTRIBUCION DE LOS PORTS DE E/S DEL PROCESADOR DE SONIDO

PORT	BIT	E/S	APLICACION	OBSERVACIONES
A	0	E	JS1 - PIN 1 *1 JS2 - PIN 1 *2	Adelante
	1	E	JS1 - PIN 2 *1 JS2 - PIN 2 *2	Atrás
	2	E	JS1 - PIN 3 *1 JS2 - PIN 3 *2	Izquierda
	3	E	JS1 - PIN 4 *1 JS2 - PIN 4 *2	Derecha
	4	E	JS1 - PIN 6 *1 JS2 - PIN 6 *2	Botón de disparo 1
	5	E	JS1 - PIN 7 *1 JS2 - PIN 7 *2	Botón de disparo 2
	6	E	Selección caracteres del teclado	Sólo versión Japonesa
	7	E	CSAR	Lectura cassette
B	0	S	JS1 - PIN 6 *3	
	1	S	JS1 - PIN 7 *3	
	2	S	JS2 - PIN 6 *3	
	3	S	JS2 - PIN 7 *3	
	4	S	JS1 - PIN 7	
	5	S	JS2 - PIN 8	
	6	S	Selección entrada Port A	Selección JS1/JS2
	7	S	Klamp (lámpara Kana)	Sólo versión japonesa

JS1 - JOYSTICK 1

JS2 - JOYSTICK 2

E - ENTRADA

S - SALIDA

\*1 - ACTIVADO CUANDO EL BIT 6 DEL PORT B ES "0"

\*2 - ACTIVADO CUANDO EL BIT 6 DEL PORT B ES "1"

\*3 - SI EL PORT B NO SE UTILIZA PARA SALIDA, ESTE BIT DEBE SER "1"

## 2. PROCESADOR DE VIDEO (VDP)

El ordenador incorpora el circuito integrado TMS 9928 A que actúa como procesador de imagen de video. Paralelamente va incorporada una memoria adicional de memoria RAM de 16 K, donde se almacena la información de las posiciones en la pantalla, junto con la definición de cada carácter.

En cada tipo de pantalla la información queda almacenada en distintas partes de la memoria. Mediante las funciones "VPEEK" y "VPOKE" podemos examinar y alterar dichas posiciones.

La función BASE(N) se utiliza para leer o escribir una dirección base de la tabla del procesador de visualización (VDP). El contenido de los registros y la dirección base de la tabla del TMS 9928 A, que es el contenido de la pantalla, podrá modificarse directamente utilizando una variable BASE y una variable VDP. Es necesario conocer adecuadamente el TMS 9928 A ya que podría alterar la visualización normal de la pantalla.

A continuación se detallan los valores posibles de la función BASE según el tipo de pantalla utilizada.

Tabla de caracteres de la pantalla	11
Tabla de caracteres de la pantalla	12
Tabla de caracteres de la pantalla	13
Tabla de caracteres de la pantalla	14
Tabla de caracteres de la pantalla	15
Tabla de caracteres de la pantalla	16
Tabla de caracteres de la pantalla	17
Tabla de caracteres de la pantalla	18
Tabla de caracteres de la pantalla	19

Valor de N	Tabla
0	Tabla de nombres de patrones del modo de texto de 40 caracteres × 24 líneas.
2	Tabla del generador de patrones del modo de texto de 40 caracteres × 24 líneas.
5	Tabla de nombres de patrones del modo de texto de 32 caracteres × 24 líneas.
6	Tabla de colores del modo de texto de 32 caracteres × 24 líneas.
7	Tabla del generador de patrones del modo de texto de 32 × 24 caracteres.
8	Tabla de atributos de figuras móviles del modo de texto de 32 × 24 caracteres.
9	Tabla de patrones de figura móvil del modo de texto de 32 × 24 caracteres.
10	Tabla de nombres de patrones del modo de gráficos de gran definición.
11	Tabla de colores del modo de gráficos de gran definición.
12	Tabla del generador de patrones del modo de gráficos de gran definición.
13	Tabla de atributos de figuras móviles del modo de gráficos de gran definición.
14	Tabla de patrones de figura móvil del modo de gráficos de gran definición.
15	Tabla de nombres de patrones del modo multicolor.
17	Tabla del generador de patrones del modo multicolor.
18	Tabla de atributos de figuras móviles del modo multicolor.
19	Tabla de patrones de figura móvil del modo multicolor.

N = 1, 3, 4, y 16 no se utilizan.

En cada tipo de pantalla (SCREEN) se manejan una serie de tablas cuyos detalles se especifican a continuación.

### SCREEN 0

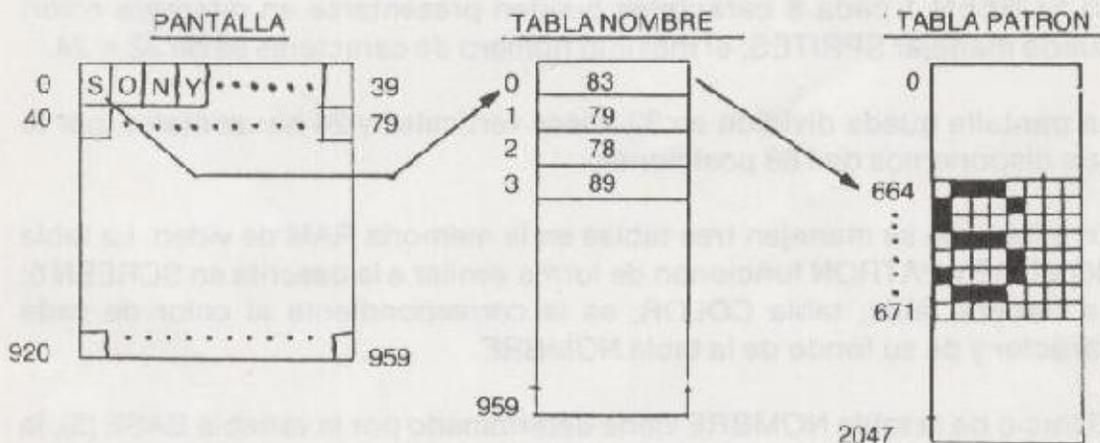
SCREEN 0 sólo puede contener caracteres (Máximo: 40 × 24); un color de pantalla; no es posible utilizar SPRITES y el color de los bordes es el de fondo.

La pantalla queda dividida en 40 líneas verticales y 24 líneas horizontales, por lo que disponemos de 960 posiciones.

Existen dos tablas para el manejo y almacenamiento de los caracteres. La tabla NOMBRE contiene el código de carácter de cada posición de pantalla. La tabla PATRON contiene el diseño de los diferentes caracteres. Cada posición de la tabla NOMBRE tiene reservados 8 octetos de la tabla PATRON.

El inicio de la tabla NOMBRE viene dado por la variable BASE (0) y el de la tabla PATRON por BASE (2).

Imaginemos que en la primera línea de la pantalla tenemos la palabra SONY:



La primera posición de la pantalla es la letra S cuyo código ASCII (83) está reflejado en la primera posición de la tabla NOMBRE. Este mismo código (83), multiplicado por ocho, da la primera posición de la tabla PATRON, donde se encuentra reflejada la forma de este carácter. La estructura de este carácter se encuentra en las 8 posiciones consecutivas de la tabla PATRON (664-671).

## EJEMPLO

Si ejecuta el siguiente ejemplo podrá comprobar el contenido de la tabla PATRON:

```
10 SCREEN 0: WIDTH 40: D$=STRING$(8, "0")
20 PRINT "SONY"
30 FOR K=0 TO 3
40 I= VPEEK (BASE (0) + K)
50 Y=I *8: Z=Y + 8: J=J + 9: T=0
60 FOR X=Y TO Z-1
70 LOCATE J-5, 7+T
80 PRINT RIGHT$(D$+BIN$(VPEEK (BASE (2) - X)), 8)
90 T=T + 1
100 NEXT X, K
```

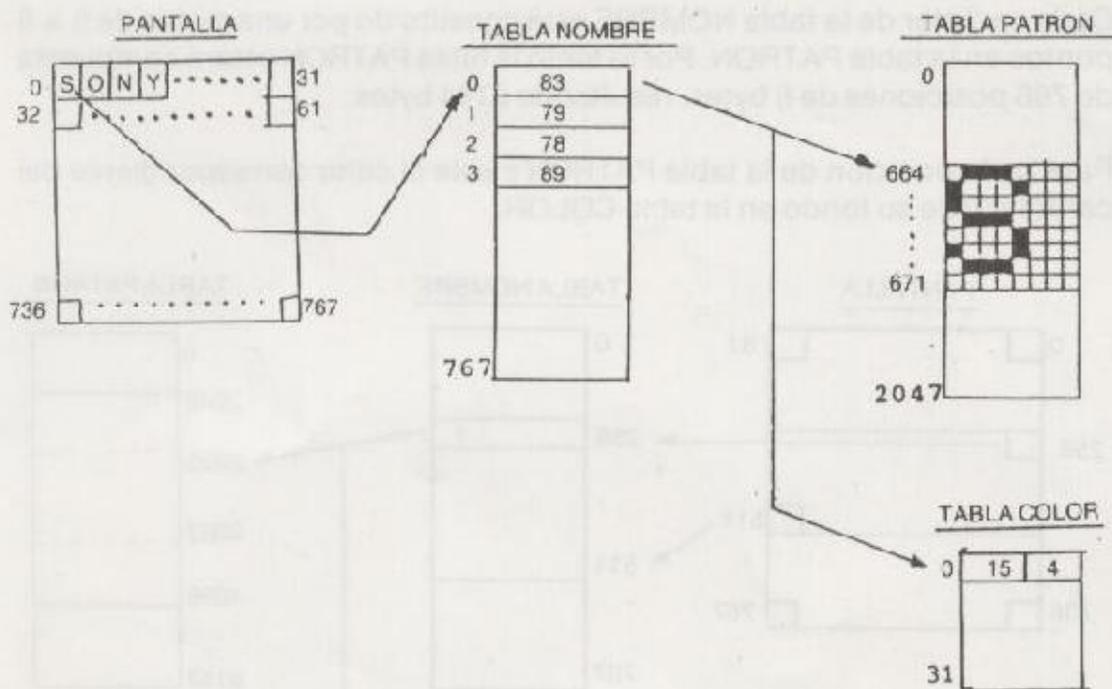
## SCREEN 1

En SCREEN 1 cada 8 caracteres pueden presentarse en diferente color; puede manejar SPRITES; el máximo número de caracteres es de  $32 \times 24$ .

La pantalla queda dividida en 32 líneas verticales y 24 horizontales, por lo que disponemos de 768 posiciones.

En este caso se manejan tres tablas en la memoria RAM de video. La tabla NOMBRE y PATRON funcionan de forma similar a la descrita en SCREEN 0. La tercera tabla, tabla COLOR, es la correspondiente al color de cada carácter y de su fondo de la tabla NOMBRE.

El inicio de la tabla NOMBRE viene determinado por la variable BASE (5), la tabla PATRON por BASE (7) y la de color por BASE (6).



Los colores de fondo y de los caracteres se pueden modificar independientemente con la sentencia COLOR. Además pueden presentarse diferentes colores a la vez.

BASE (6) indica el principio de la tabla COLOR de 32 bytes. Cada byte está dividido en dos cuartetos. El primer cuarteto indica el color de los ocho primeros caracteres consecutivos de la pantalla y el segundo cuarteto el color de fondo.

## SCREEN 2

En SCREEN 2 pueden presentarse gráficos de alta resolución y SPRITES; texto en pantalla gráfica; 256 × 192 puntos de resolución y cada 8 puntos tiene su propio color de frente y fondo.

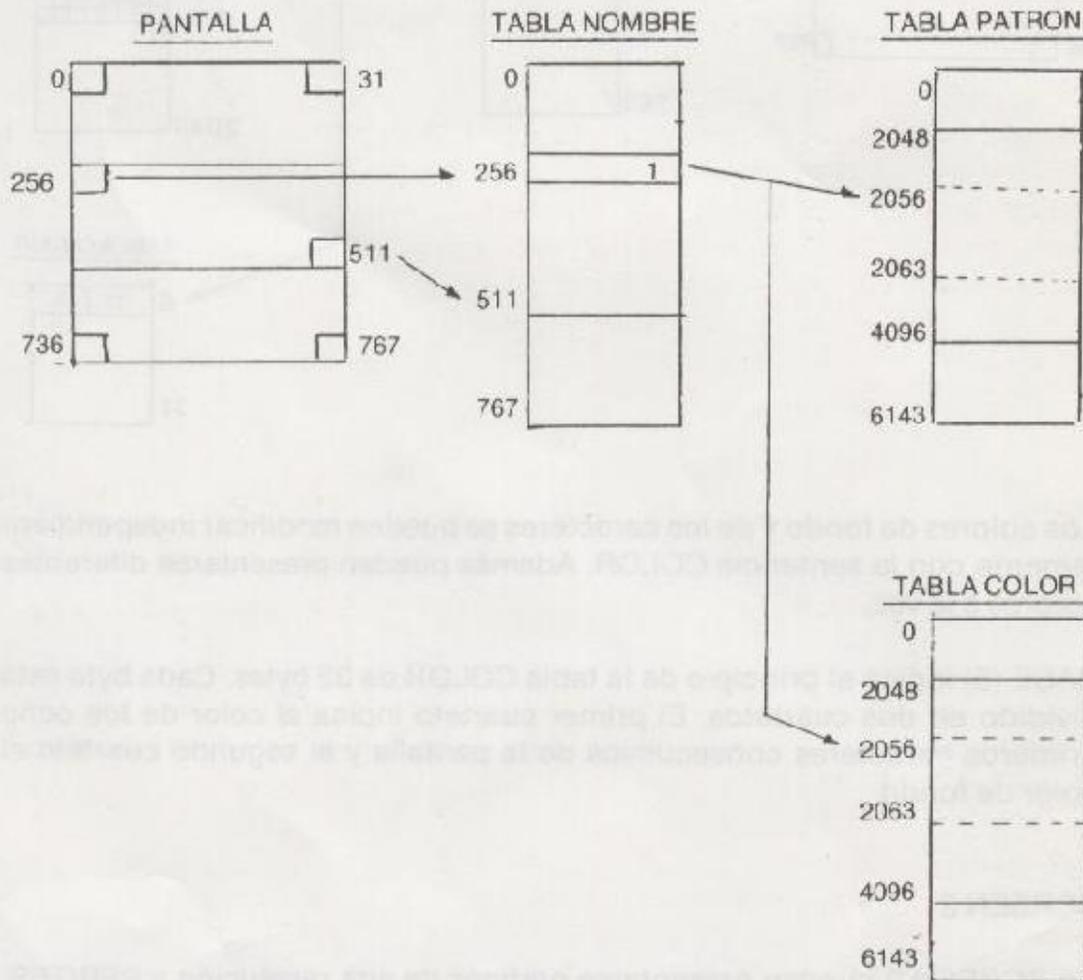
La pantalla queda dividida en 24 líneas horizontales y 32 verticales configurando un total de 768 posiciones de pantalla. Cada posición está subdividida en una malla de 8×8 puntos.

Se conserva el número de tablas, aunque varía su longitud. La tabla NOMBRE contiene el código de carácter de cada posición correlativa de la pantalla.

El código de carácter que figura en la tabla NOMBRE señala la posición del tipo y color del carácter en la tabla PATRON y tabla COLOR, respectivamente.

Cada carácter de la tabla NOMBRE está constituido por una matriz de  $8 \times 8$  puntos en la tabla PATRON. Por lo tanto la tabla PATRON estará compuesta de 768 posiciones de 8 bytes, resultando 6144 bytes.

Para cada posición de la tabla PATRON existe el color correspondiente del carácter y de su fondo en la tabla COLOR.



El inicio de la tabla NOMBRE viene determinado por la variable BASE (10), la tabla PATRON por BASE (12) y la tabla COLOR por BASE (11).

### SCREEN 3

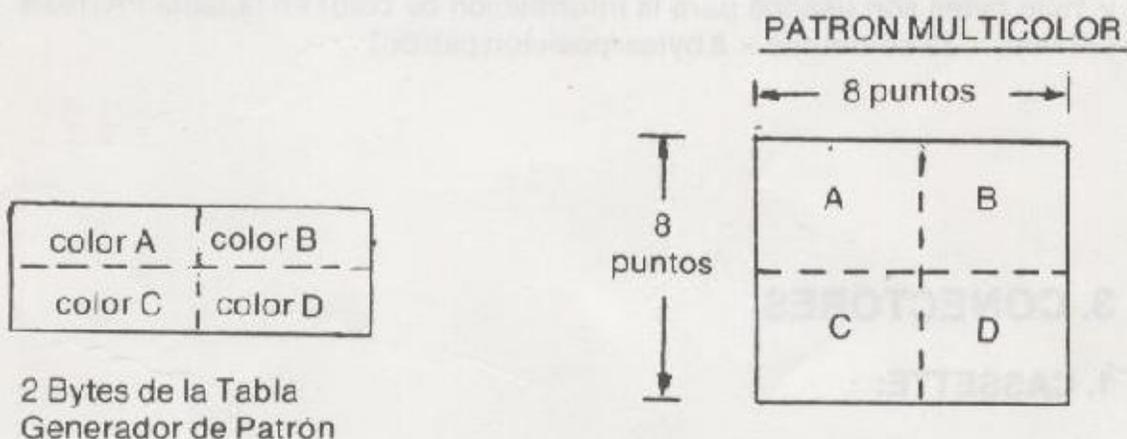
En SCREEN 3 pueden presentarse gráficos de baja resolución y texto en pantalla gráfica en modo multicolor.

Este modo proporciona una definición de  $64 \times 48$  bloques de color. Cada bloque contiene  $4 \times 4$  puntos. El color de cada uno de los cuatro puntos

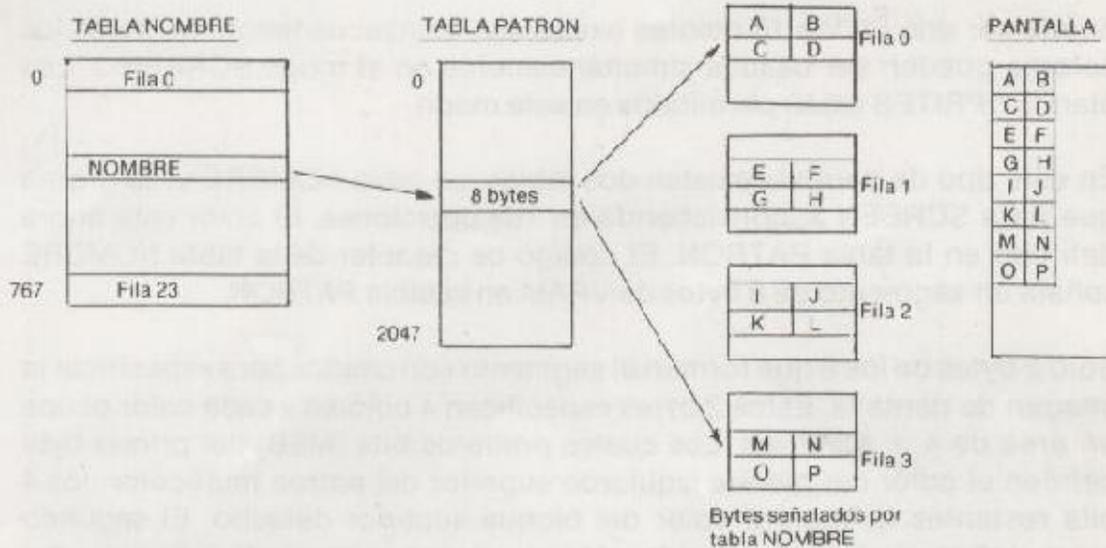
puede ser uno de los 16 colores existentes. Consecuentemente, todos los colores pueden ser usados simultáneamente en el modo SCREEN 3. Los planos SPRITES están permitidos en este modo.

En este tipo de pantalla existen dos tablas. La tabla NOMBRE es la misma que para SCREEN 2, consistiendo en 768 posiciones. El color está ahora definido en la tabla PATRON. El código de carácter de la tabla NOMBRE señala un segmento de 8 bytes de VRAM en la tabla PATRON.

Sólo 2 bytes de los 8 que forman el segmento son usados para especificar la imagen de pantalla. Estos 2 bytes especifican 4 colores y cada color ocupa un área de  $4 \times 4$  puntos. Los cuatro primeros bits (MSB) del primer byte definen el color del bloque izquierdo superior del patrón multicolor; los 4 bits restantes definen el color del bloque superior derecho. El segundo bytes define, de forma parecida, el bloque inferior izquierdo y derecho del patrón multicolor. Los 2 bytes crean un mapa de  $8 \times 8$  puntos de patrón multicolor.



La posición de los 2 bytes dentro del segmento de 8 bytes señalado por el código de carácter de la tabla NOMBRE depende de la posición que aparece el carácter en la pantalla. Para nombres en la fila superior (fila 0), los 2 bytes son los 2 primeros dentro de los grupos de segmentos de 8 bytes señalados por el código de carácter de la tabla NOMBRE. La siguiente fila de caracteres (fila 1) utiliza los bytes 3 y 4 de los segmentos de 8 bytes. La siguiente fila utiliza los bytes 5 y 6 mientras que la última fila utiliza los bytes 7 y 8. Esta serie se repite para el resto de pantalla.



Cuando se utiliza este modo, 768 bytes son utilizados por la tabla NOMBRE y 1536 bytes son usados para la información de color en la tabla PATRON (24 filas  $\times$  32 columnas  $\times$  8 bytes/posición patrón).

### 3. CONECTORES

#### 1. CASSETTE:

PIN	SEÑAL	E/S	CONECTOR
1	MASA	—	
2	MASA	—	
3	MASA	—	
4	SALIDA	S	
5	ENTRADA	E	
6	REMOTO +	S	
7	REMOTO —	S	
8	MASA	—	

E = ENTRADA  
S = SALIDA

## 2. AUDIO/VIDEO

PIN	SEÑAL	E/S	CONECTOR
1	+ 12V	—	
2	MASA	—	
3	+ 12V	—	
4	VIDEO	S	
5	AUDIO	S	
6	AUDIO	S	

## 3. JOYSTICK

PIN	SEÑAL	E/S	CONECTOR
1	ADELANTE	E	
2	ATRAS	E	
3	IZQUIERDA	E	
4	DERECHA	E	
5	+ 5V.	—	
6	B. DISPARO 1	E/S	
7	B. DISPARO 2	S	
8	SALIDA	S	
9	MASA	—	

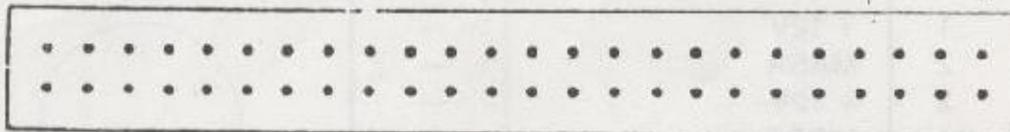
## 4. IMPRESORA:

PIN	SEÑAL	PIN	SEÑAL	CONECTOR
1	PSTB	8	DATO 6	
2	DATO 0	9	DATO 7	
3	DATO 1	10	SIN CONEX.	
4	DATO 2	11	BUSY	
5	DATO 3	12	SIN CONEX.	
6	DATO 4	13	SIN CONEX.	
7	DATO 5	14	MASA	

## 5. CARTUCHO:

Pin 49

1



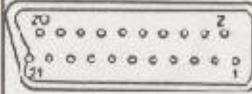
Pin 50

2

PIN	SEÑAL	E/S	PIN	SEÑAL	E/S
1	CS1	S	2	CS2	S
3	CS12	S	4	SLTSL	S
5	Reservado	—	6	RFSH	S
7	WAIT	E	8	INT	E
9	M1	S	10	BUSDIR	E
11	IORQ	S	12	MERQ	S
13	WR	S	14	RD	S
15	RESET	S	16	Reservado	—
17	A9	S	18	A15	S
19	A11	S	20	A10	S
21	A7	S	22	A6	S
23	A12	S	24	A8	S
25	A14	S	26	A13	S
27	A1	S	28	A0	S
29	A3	S	30	A2	S
31	A5	S	32	A4	S
33	D1	E/S	34	D0	E/S
35	D3	E/S	36	D2	E/S
37	D5	E/S	38	D4	E/S
39	D7	E/S	40	D6	E/S
41	Masa	—	42	Reloj	S
43	Masa	—	44	SW1	—
45	+ 5V	—	46	SW2	—
47	+ 5V	—	48	- 12V	—
49	SOUNDIN	E	50	- 12V	—

PIN	NOMBRE	CONTENIDO
1	CS1	Señal de selección de direcciones ROM 4000/7FFF
2	CS2	Señal de selección de direcciones ROM 8000/BFFF
3	CS12	Señal de selección de direcciones ROM 4000/BFFF (para 256 K ROM)
4	SŁTSL	Señal de selección de SLOT
5	Reservado	Línea de señal reservada (uso inhibido)
6	RFSH	Ciclo de señal de retresco
7	WAIT	Señal de requerimiento de espera de la CPU
8	INT	Señal de requerimiento de interrupción de la CPU
9	M1	Señal de ciclo de fetch de CPU
10	BUSDIR	Esta señal controla de dirección de buffer del bus de datos externos. Los cartuchos son seleccionados y el nivel "L" es sacado de cada cartucho en el momento de transmisión de datos.
11	IORQ	Señal de requerimiento E/S
12	MERQ	Señal de requerimiento de memoria.
13	WR	Señal tiempo escritura
14	RD	Señal de tiempo de lectura
15	RESET	Señal de RESET del sistema.
16	RESERVADO	Línea de señal reservada (uso inhibido)
17	32 AO A15	Señales del bus de direcciones.
33	40 DO D7	Señales del bus de datos.
41	MASA	Señal de Masa.
42	RELOJ	Reloj CPU 3.58 MHZ.
43	MASA	Señal de Masa.
44, 46	SW1, SW2	Para protección de Inserción/Ordenación.
45, 47	+ 5V	Tensión - 5V.
48	+ 12 V	Tensión - 12V.
49	SOUNDIN	Señal de entrada de sonido (- 5 dbm).
50	- 12 V	Tensión 12 V.

## 6. RGB:

PIN	SEÑAL	PIN	SEÑAL	CONECTOR
1	SAL. AUDIO D	12		
A 2	ENT. AUDIO D	13	MASA	
3	SAL. AUDIO I	14		
4	MASA	15	COLOR ROJO	
5	MASA	16	BLANKING	
Δ 6	ENT. AUDIO I	17	MASA	
7	COLOR AZUL	18	MASA	
8	—	19	SAL. VIDEO	
9	MASA	20	—	
10	—	21	—	
11	COLOR VERDE			

19 : masa. video  
20 : ent. video



## ANEXO 4: PROGRAMAS

### PROGRAMA DEMOSTRACION DE LOS COMANDOS GRAFICOS UTILIZADOS EN EL BASIC MSX

```
1 REM ** COMANDOS GRAFICOS **
10 COLOR 15,1,1
15 *** ALTA RESOLUCION -->SCREEN 2
20 SCREEN 2
25 *** PUNTO ***
30 PSET(32,32),15
35 *** LINEA ***
40 LINE(76,32)-(116,32),8
45 *** RECTANGULO ***
50 LINE(135,12)-(180,52),12,B
55 *** PINTAR UN RECTANGULO ***
60 LINE(204,12)-(244,52),8,BF
65 *** CIRCULO ***
70 CIRCLE(32,96),20,5
75 *** PINTAR UN CIRCULO ***
80 CIRCLE(96,96),20,4
90 PAINT(96,96),4
95 *** ELIPSE HORIZONTAL ***
100 CIRCLE(160,96),20,15,...5
105 ***PINTAR UNA ELIPSE HORIZONTAL***
110 CIRCLE(224,96),20,13,...5
120 PAINT(224,96),13
125 *** ELIPSE VERTICAL ***
130 CIRCLE(32,160),20,2,..2
```

```

135 *** PINTAR UNA ELIPSE VERTICAL ***
140 CIRCLE(96,160),20,11,,,2
150 PAINT(96,160),11
155 *** MACRO LENGUAJE GRAFICO ***
160 DRAW"c2bm140,162 r10e20f5r10e10f5e10
r5e5f10r5f10"
170 GOTO 170

```

### PROGRAMA DE GRAFICOS: SIMULACION REBOTES DE UNA PELOTA

```

10 ***REBOTES DE UNA PELOTA***
20 SCREEN 2,COLOR 15,1,1
30 CLS:A=RND(-TIME):T=INT(RND(1)*14+2):O
PEN"GRP:"FOR OUTPUT AS#1:PRESET(70,10):C
OLOR T:PRINT#1,"REBOTES DE UNA PELOTA"
40 X=10:Y=10:A=4:V=0:C=0
50 CIRCLE(X,Y),5,C,,,1.2
60 ON INTERVAL=5 GOSUB 100
70 INTERVAL ON
80 IF X<256 THEN 80
90 INTERVAL OFF:CLOSE:GOTO 30
100 CIRCLE(X,Y),5,T,,,1.2
110 V=V+A
120 X=X+5:Y=Y+V
130 IF Y>180 THEN Y=180*2-Y:V=-V*.8:BEEP

140 CIRCLE(X,Y),5,T
150 RETURN

```

### PROGRAMA DE GRAFICOS: CUADROS ABSTRACTOS

```

10 REM *** CUADROS ABSTRACTOS ***
20 SCREEN 2:COLOR 15,1,1:CLS
22 A=RND(-TIME)
25 FOR I=1 TO 75

```

```

27 C=INT(RND(1)*14)+2
30 X1=INT(RND(1)*255)
40 Y1=INT(RND(1)*192)
50 X2=INT(RND(1)*255)
60 Y2=INT(RND(1)*192)
70 LINE(X1,Y1)-(X2,Y2),C,BF
80 NEXT
90 FOR I=1 TO 2000:NEXT:GOTO 20

```

### PROGRAMA DE GRAFICOS: CIRCULOS DE COLORES

```

10 COLOR 15,1,1
20 SCREEN2:A=RND(-TIME)
30 FOR X=0 TO 2*3.14159 STEP .104719666#
40 CIRCLE(124+40*COS(X),106+40*SIN(X)),2
0,INT(RND(1)*15)+1,,1
50 NEXT
60 GOTO 60

```

### PROGRAMA DE GRAFICOS: FLORES

```

10 '** GRAFICOS ALEATORIOS **
20 COLOR 15,1,1:SCREEN 2:A=RND(-TIME):
30 R1=5:R2=20:Z=3.14/180
40 X0=INT(256*RND(1)):Y0=INT(192*RND(1))
:C=INT(14*RND(1))+2
50 FOR T=0 TO 360 STEP 20
60 S=T*Z
70 Y=SIN(S):X=COS(S)
80 LINE(R1*X+X0,R1*Y+Y0)-(R2*X+X0,R2*Y+Y
0),C
90 NEXT T
100 GOTO 40

```

## PROGRAMA DE GRAFICOS: DIBUJO DE CIRCULOS DE COLORES ALEATORIOS EN FORMA SENOIDAL

```
5 *** GRAFICOS ALEATORIOS ***
10 COLOR 15,1,1
20 SCREEN2:C=2:A=RND(-TIME)
30 FOR X=0 TO 6.28 STEP.157
40 COLOR C:C=C+1:IF C=16 THEN C=2
50 Y=SIN(X)*80+96:A=INT(RND(1)*96)
60 Z=COS(X+(3.1415/2))*80+96
70 CIRCLE(X*30+20,Y),12,....5:PLAY"N=A:"

80 CIRCLE(X*30-20,Z),12,....5
90 NEXT
100 GOTO 32
```

## PROGRAMA DE GRAFICOS: 'ESTRELLAS'

```
10 REM $ GRAFICOS:PSET $
20 REM 'CIELO ESTRELLADO'
30 A=RND(-TIME):COLOR 15,1,1:SCREEN 2,1:
FOR I=0 TO 200
40 X=INT(RND(1)*255)+1
50 Y=INT(RND(1)*192)+1
60 PSET(X,Y),INT(RND(1)*14)+2
70 NEXT I
80 FOR I=1 TO 2000 :NEXT I
90 SPRITE$(0)=CHR$(&B00000000)+CHR$(&B20
000000)+CHR$(&B00100100)+CHR$(&B00111120
)+CHR$(&B01011010)+CHR$(&B10011001)+CHR$
(&B01111110)+CHR$(&B20100100)
100 PUT SPRITE 2,(120,90)
110 FOR Q=1 TO 200:X=INT(RND(1)*20)*(-1)
^Y:Y=INT(RND(1)*20)*(-1)^X
120 PUT SPRITE 0,STEP(X,Y),5:BEEP
130 FOR T=1 TO 70:NEXT
140 NEXT Q:FOR I=1 TO 2000:NEXT:CLS:GOTO
30
```

## PROGRAMA DEMOSTRACION DE GRAFICOS. 1.

```
10 REM **** GRAFICOS ****
20 ' INICIALIZACION
30 XC=128:YC=96
40 R=8
50 SP=6
60 RATIO=1.5
70 RX=1
80 RY=1
90 PI=3.14159:FLAG=1
100 XB=0:YB=0:XE=0:YE=0:XM=0:YM=0
110 DIM X(5),Y(5),A(5)
120 A(0)=0:A(1)=-15:A(2)=0:A(3)=-15:A(4)
=0:A(5)=-15
130 FOR NO=0 TO 5
140 A(NO)=A(NO)/180*PI
150 NEXT NO
160 C=2
170 ' PROGRAMA PRINCIPAL
180 SCREEN 2:COLOR 7,1,1:CLS
190 UW=0:FOR ANGLE=0 TO 360 STEP SP
200 UW=UW+1 :C=5
210 AG=ANGLE/180*PI
220 RB=R
230 FOR NO=0 TO 5
240 X(NO)=RB*RATIO*COS(AG+A(NO))*RX
250 Y(NO)=RB*RATIO*SIN(AG+A(NO))*RY
260 RB=RB*RATIO
270 NEXT NO
280 FOR NO=0 TO 4
290 LINE(X(NO)+XC,Y(NO)+YC)-(X(NO+1)+XC,
Y(NO+1)+YC),C
300 NEXT NO
310 IF XB<>0 AND FLAG=-1 THEN LINE(XB+XC
,YB+YC)-(X(0)+XC,Y(0)+YC),C
320 IF XE<>0 AND FLAG=-1 THEN LINE(XE+XC
,YE+YC)-(X(5)+XC,Y(5)+YC),C
330 IF XM<>0 AND FLAG=-1 THEN PAINT((XM+
X(4)+XC*2)/2,(YM+Y(4)+YC*2)/2),C
340 XB=X(0):YB=Y(0)
350 XE=X(5):YE=Y(5)
360 XM=X(4):YM=Y(4)
```

```

370 FLAG=-FLAG
380 IF UW=1 THEN 392 ELSE UW=0
390 NEXT ANGLE
400 FOR I=1 TO 5000:NEXT I: CLEAR:CLS:GOT
0 10

```

## PROGRAMA DEMOSTRACION DE GRAFICOS. 2.

```

10 REM **** GRAFICOS ****
20 * INICIALIZACION
30 DIM R(9),S(9)
40 XS=3
50 XF=2
60 YW=50
70 R(1)=48:R(2)=10:R(3)=5:R(4)=5:R(5)=5:
R(6)=5:R(7)=10:R(8)=48:R(9)=0:S(0)=0
80 FOR NO=1 TO 8
90 S(NO)=S(NO-1)+R(NO)
100 NEXT NO
110 PI=3.14159
120 *PROGRAMA PRINCIPAL
130 SCREEN 2:COLOR 8,1,1:CLS
140 FOR X=36 TO 222 STEP XS
150 Y=YW* SIN(XF*X/180*PI)
160 FOR NO=1 TO 7 STEP 2
170 CIRCLE(X-R(NO)*COS(PI/4), (S(NO-1))*2+
R(NO))*SIN(PI/4)+Y),R(NO),,PI*7/4,PI/4
180 CIRCLE(X+R(NO+1)*COS(PI/4), (S(NO))*2+
R(NO+1))*SIN(PI/4)+Y),R(NO+1),,PI*3/4,PI
*5/4
190 NEXT NO
200 NEXT X
210 FOR I=1 TO 5000:NEXT I: CLEAR:CLS:GOT
0 20

```

**PROGRAMA DE GRAFICOS: 'SPRITES'  
MOVIMIENTO DEL CAÑON → CURSORES  
DISPAROS → BARRA DE ESPACIO**

```
5 SCREEN 1,0:KEY OFF:COLOR15,4,4:A=RND(-  
TIME)  
20 FOR I=0 TO 2  
30 GOSUB 3010  
40 SPRITE$(I)=SP$  
50 NEXT I  
70 ON STRIG GOSUB 1010:STRIG(0)ON  
80 ON SPRITE GOSUB 2010  
90 X0=0:Y0=30:X1=120:Y1=170  
100 PUT SPRITE 1,(X1,Y1),15,1  
120 ON (STICK(0)+2)/4 GOSUB 200,300  
130 X0=X0+INT(RND(1)*5+1)  
140 Y0=Y0+INT(RND(1)*11-5)  
150 PUT SPRITE 0,(X0,Y0),15,0  
160 GOTO 100  
200 X1=X1+2:RETURN  
300 X1=X1-2:RETURN  
1010 SOUND 6,15:SOUND 7,7  
1020 SOUND 8,16:SOUND 9,16  
1030 SOUND 10,16:SOUND 11,0  
1040 SOUND 12,10:SOUND 13,1  
1050 STRIG(0) ON:SPRITE ON  
1060 X2=X1  
1070 FOR Y2=Y1-8 TO -8 STEP -1  
1080 PUT SPRITE2,(X2,Y2),15,2  
1090 NEXT Y2  
1100 RETURN  
2010 SOUND 0,65:SOUND 1,15  
2020 SOUND 2,97:SOUND 3,15  
2030 SOUND 4,162:SOUND 5,15  
2040 SOUND 6,15:SOUND 7,21  
2050 SOUND 8,31:SOUND 9,31  
2060 SOUND 10,31:SOUND 11,228  
2070 SOUND 12,37:SOUND 13,1  
2080 PUT SPRITE0,(X0,Y0),8,0  
2090 LOCATE 10,12:PRINT"GAME OVER"
```

```

2100 FOR I=1 TO 1000:NEXT:RESTORE:SPRITE
OFF:CLEAR:GOTO 5
3010 SP$=""
3020 FOR J=1 TO 8
3030 READ D$
3040 SP$=SP$+CHR$(VAL("&B"+D$))
3050 NEXT J
3060 RETURN
4010 DATA 00011000
4020 DATA 00111100
4030 DATA 01111110
4040 DATA 11011011
4050 DATA 11011011
4060 DATA 11111111
4070 DATA 00100100
4080 DATA 01000010
4100 DATA 00010000
4110 DATA 00111000
4120 DATA 00111000
4130 DATA 01111100
4140 DATA 11111111
4150 DATA 10101010
4160 DATA 11111110
4170 DATA 11000110
4190 DATA 00010000
4200 DATA 00010000
4210 DATA 00020000
4220 DATA 00020000
4230 DATA 00020000
4240 DATA 00020000
4250 DATA 00020000
4260 DATA 00020000

```

## PROGRAMA DE MUSICA: ORGANO MUSICAL

```
10 REM *** ORGANO MUSICAL ***
20 '
30 '=====

40 ' TECLAS:.QWERTYU --> DO-SI(OCTAVA 2)
50 '      .I      --> DO   (OCTAVA 3)
60 '      .ASDFGHJ --> DO-SI(OCTAVA 4)
70 '      .K      --> DO   (OCTAVA 5)
80 '      .ZXCVBNM --> DO-SI(OCTAVA 6)
90 '      .,(COMA) --> DO   (OCTAVA 7)

100 '=====
=
110 '
120 SCREEN 2:CLS:COLOR 8,1,1
130 OPEN"GRP:"FOR OUTPUT AS #1
140 A$="U20R20D20L20"
150 FOR X=20 TO 190 STEP 25
160 PRESET(X,40):DRAW "C5XA$;"
170 NEXT
180 PRESET(20,70):DRAW "C5XA$;"
190 FOR X=20 TO 190 STEP 25
200 PRESET(X,100):DRAW "C5XA$;"
210 NEXT
220 PRESET(20,130):DRAW"C5XA$;"
230 FOR X=20 TO 190 STEP 25
240 PRESET(X,160):DRAW"C5XA$;"
250 NEXT
260 PRESET(20,190):DRAW"C5XA$;"
270 FOR X=27 TO 190 STEP 25
280 READ A$
290 PRESET(X,27):PRINT#1,A$
300 NEXT
310 PRESET(27,57):READ A$:PRINT#1,A$
320 FOR X=27 TO 190 STEP 25
330 READ A$
```

```

340 PRESET(X,87):PRINT#1,A$
350 NEXT
360 PRESET(27,117):READ A$:PRINT#1,A$
370 FOR X=27 TO 190 STEP 25
380 READ A$
390 PRESET(X,147):PRINT#1,A$
400 NEXT
410 PRESET(27,177):READ A$:PRINT#1,A$
420 COLOR 14
430 FOR X=23 TO 185 STEP 25
440 READ A$
450 PRESET(X,8):PRINT#1,A$
460 NEXT
470 PRESET(200,8):PRINT#1,"OCTAVA"
480 COLOR 8
490 FOR Y=27 TO 177 STEP 30
500 PRESET(210,Y):READ A:PRINT#1,A
510 NEXT
520 PRESET(50,57):COLOR 2:PRINT#1," ORGA
NO MUSICAL"
530 PRESET(70,117):PRINT#1," ♪ ♫ ♬ ♪"
540 GOSUB 900
550 PLAY"L15V15"
560 A$=INKEY$:IF A$="" GOTO 560
570 IF A$="Q" THEN PLAY"O2C"
580 IF A$="W" THEN PLAY"O2D"
590 IF A$="E" THEN PLAY"O2E"
600 IF A$="R" THEN PLAY"O2F"
610 IF A$="T" THEN PLAY"O2G"
620 IF A$="Y" THEN PLAY"O2A"
630 IF A$="U" THEN PLAY"O2B"
640 IF A$="I" THEN PLAY"O3C"
650 IF A$="A" THEN PLAY"O4C"
660 IF A$="S" THEN PLAY"O4D"
670 IF A$="D" THEN PLAY"O4E"
680 IF A$="F" THEN PLAY"O4F"
690 IF A$="G" THEN PLAY"O4G"
700 IF A$="H" THEN PLAY"O4A"
710 IF A$="J" THEN PLAY"O4B"
720 IF A$="K" THEN PLAY"O5C"
730 IF A$="Z" THEN PLAY"O6C"
740 IF A$="X" THEN PLAY"O6D"
750 IF A$="C" THEN PLAY"O6E"
760 IF A$="V" THEN PLAY"O6F"

```

```

770 IF A$="B" THEN PLAY"06G"
780 IF A$="N" THEN PLAY"06A"
790 IF A$="M" THEN PLAY"06B"
800 IF A$="," THEN PLAY"07C"
810 GOTO 560
820 DATA Q,W,E,R,T,Y,U
830 DATA I
840 DATA A,S,D,F,G,H,J
850 DATA K
860 DATA Z,X,C,V,B,N,M
870 DATA ","
880 DATA DO,RE,MI,FA,SOL,LA,SI
890 DATA 1,2,3,4,5,6
900 REM **** MUSICA INICIAL ****
910 '
920 A$="L806C4.ECEG16L64CFAL807C06BAGG5.
.O5L64GB06DL8GF005BGB06DFECAGG4"
930 B$="05L8CGEG"
940 C$="05CAFA"
950 D$="04G05FDF"
960 F$=B$+B$+C$+B$+D$+D$+B$+B$
970 T$="L806C4.ECEG16L64CFAL807C06BAGG5.
.O5L64GB06DL8GF005BGB06D05B06C05G06ECC2"

980 G$=B$+B$+C$+B$+D$+D$+B$+B$
990 PLAY"S1M3000T120","S8M500T120":PLAY
A$,F$:PLAY T$,G$
1000 RETURN

```

### PROGRAMA DE MUSICA: HIMNO

```

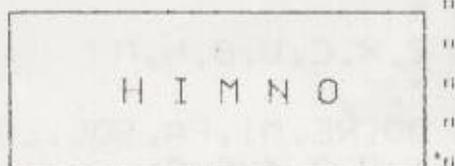
10 REM **** PROGRAMA MUSICAL ****
20 REM          H I M N O
30 COLOR 4,14,14:CLS:GOSUB 100:CLEAR:PLA
Y"U12","U12","U12"
40 U1$='T10806L4CCDC5G.06C8DEEFE.D8CE8D8
C05GC6C.D8E8F8GGGG.F8FFFFFF.E8DEL8FEDCL4E
.F8GL12AGFL4EDCR4R4 R20"
50 U2$="T10805L4EGAD.G8GGAAGBAFEDE.R8R4G
GGG.G8GGGGG.G8GGL8AGFEL4G.F8GR4GFER4R4 R
20"

```

```

60 U3$="T10804L4DEFG.A8B05C04AFGG#AFG03G
04C03CR404CEG05C04CR403CB04DG03GR404C05C
CC.04A8E78D8G03G04C03GC R20"
70 PLAY U1$,U2$,U3$
80 FOR F=1 TO 8000:NEXT F
90 GOTO 70
120 SCREEN 0:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT:PRINT:PRINT
110 PRINT"
120 PRINT"
130 PRINT"
140 PRINT"
150 PRINT"
160 RETURN

```



### PROGRAMA DE MUSICA: 'ROCK'

```

10 REM ***** MUSICA *****
20 REM EFECTOS ESPECIALES
30 REM ** ROCK **
40 CLS:KEY OFF:CLEAR:COLOR 1,INT(RND(1)*
15)+2
50 LOCATE 14,17:PRINT"R O C K"
60 B=500:PLAY"U12"
70 A$="T23003EG#B04C#DC#03BG#"
80 B$="T23003A04C#EF#GF#EC#"
90 C$="T23003B04D#F#G#AG#F#D#"
100 FOR A=3 TO 15
110 IF A=9 OR A=5 OR A=6 OR A=7 OR A=15
THEN 150
120 PLAY"M=B;":PLAY"S=A;"
130 FOR I=1 TO 2:PLAY A$:COLOR,INT(RND(1)
)*14)+2:NEXT
140 PLAY B$:COLOR,INT(RND(1)*14)+2:PLAY
A$:COLOR,INT(RND(1)*14)+2:PLAY C$:COLOR,
INT(RND(1)*14)+2:PLAY B$:COLOR,INT(RND(1)
)*14)+2
150 NEXT A:GOTO 100

```

**PROGRAMA PARA HACER UNA COPIA DE LA PANTALLA  
(SCREEN 0) POR IMPRESORA**

```
10 REM *** HARD-COPY ->SCREEN 0 ***
20 '
30 FOR K=0 TO 960
40 A=UPEEK(BASE[0]+K)
50 LPRINT CHR$(A);
60 IF L=39 THEN 80 ELSE IF L<39 THEN L=L
+1
70 NEXTK:END
80 LPRINT:L=0:GOTO 70
```

**PROGRAMA PARA HACER UNA COPIA DE LA PANTALLA  
(SCREEN 1) POR IMPRESORA**

```
10 REM *** HARD-COPY ->SCREEN 1 ***
20 '
30 FOR K=0 TO 767
40 A=UPEEK(BASE[5]+K)
50 LPRINT CHR$(A);
60 IF L=31 THEN 80 ELSE IF L<31 THEN L=L
+1
70 NEXTK:END
80 LPRINT:L=0:GOTO 70
```

**PROGRAMA PARA HACER UNA COPIA DE LA PANTALLA  
(SCREEN 2/3) CON EL PLOTTER PRN-C41 DE SONY.**

```
10 ' ***** HARD-COPY *****
20 '
30 ' PLOTTER PRN-C41 SONY
40 '
50 ' SCREEN 2/SCREEN 3
60 '
70 '
```

```

80 COLCR 4,4
90 LPRINT
100 LPRINT CHR$(("&H1B")+ "#")
110 FOR Y=0 TO 192
120 FOR X=0 TO 250
130 IF POINT(X,Y)=4 THEN LPRINT "R3,0":GO
TO 160
140 LPRINT "J0,3,3,0,0,-3,-3,0"
150 LPRINT "R3,0"
160 NEXT X
170 LPRINT "R-753,-3"
180 NEXT Y

```

**PROGRAMA PARA UTILIZAR CON EL PLOTTER PRN-C41  
DE SONY: 'CIRCULO'**

```

10 * *** PLOTTER PRN-C41 ***
20 *           SONY
30 *           C I R C U L O
40 LPRINT CHR$(27):"# "
50 LPRINT "R125,-125"
50 LPRINT "I"
70 LPRINT "M80,0"
80 P=3.14159
90 A=1:B=1
100 FOR R=0 TO 360 STEP 5
110 S=R/180*P
120 SI=INT(SIN(A*S)*80)
130 CO=INT(COS(B*S)*80)
140 LPRINT "D":CO:",";SI
150 NEXT
160 LPRINT "R0,-125"
170 LPRINT "A"

```

## PROGRAMA PARA UTILIZAR CON EL PLOTTER PRN-C41 DE SONY: 'CUADRICULA'

```
10 ' *** PLOTTER PRN-C41 ***
20 '           SONY
30 '   CUADRICULA DE COLORES
40 LPRINT
50 LPRINT CHR$(27):"#"
60 LPRINT"I" ;C=0
70 FOR I=1 TO 20
80 LPRINT"C" ;C
90 IF I MOD 5=0 THEN C=C+1
100 IF C>3 THEN C=0
110 LPRINT"J198,0"
120 LPRINT"R0,-5"
130 LPRINT"J-198,0"
140 LPRINT"R0,-5"
150 NEXT
160 LPRINT"R0,5"
170 FOR I=1 TO 20
180 LPRINT"C" ;C
190 IF I MOD 5=0 THEN C=C+1
200 IF C>3 THEN C=0
210 LPRINT"J0,198"
220 LPRINT"R5,0"
230 LPRINT"J0,-198"
240 LPRINT"R5,0"
250 NEXT
260 LPRINT:LPRINT
270 LPRINT"A"
```

## JUEGO: LABERINTO

```
5 L=2
6 CLEAR:RESTORE:COLOR 7,1,1
10 'LABERINTO'
11 CLS:SCREEN 1:PRINT:PRINT:PRINT:PRINT
12 PRINT"
13 PRINT"          LABERINTO
14 PRINT"
```

```

15 FOR I=1 TO 1000
16 NEXT I
20 ON ERROR GOTO 470
30 DEFINT A-Z:DEF FNE(X,Y)=((X=WX-1) AND
(Y=WY-1))
40 DIM DX(3),DY(3):FOR I=0 TO 3:READ DX(
I),DY(I):NEXT
50 DATA 2,0,0,2,-2,0,-2
60 LOCATE 7,8:PRINT"NIVEL ? (1-3)":A$=I
NKEY$:L=VAL(A$):IF L<1 OR L>3 GOTO 60
70 FOR I=1 TO L:READ WX:NEXT :WY=WX
80 DATA 5,8,12
90 FOR I=1 TO VAL(RIGHT$(STR$(TIME),2)):
X=RND(1):NEXT
91 PRINT:PRINT:PRINT:PRINT
100 PRINT"ESTAS EN UN LABERINTO. DEBES":
PRINT:PRINT" BUSCAR LA SALIDA":PRINT:
PRINT:PRINT:PRINT" UTILIZA LOS CURSORES
PARA":PRINT:PRINT" MOVERTE POR LOS PAS
ILLOS":KX=WX+1:KY=WY+1:WX=WX*2:WY=WY*2
110 DIM M(WX,WY)
120 FOR I=0 TO WX:M(0,I)=1 :M(I,0)=1 :M(
WX,I)=1:M(I,WY)=1:NEXT
130 C=(KX-2)*(KY-2):CA=C-4
140 X=INT(RND(1)*KX)*2:Y=INT(RND(1)*KY)*
2 :IF M(X,Y)=0 GOTO 140
150 GOSUB 260:IF C<CA THEN IF CN GOTO 1
50
160 IF C<>0 GOTO 140
170 *JUEGO
180 SCREEN 2:XM=1:YM=1:DR=0
190 GOSUB 320 *LINE
200 IF FNE(XM,YM) THEN OPEN "GRP:"FOR O
UTPUT AS #1:PSET(90,10):PRINT#1,"S A L I
D A":GOSUB 500:FOR QW=1 TO 1500:NEXT QW
:SCREEN 1:LOCATE 1,10:PRINT" HAS ENCO
NTRADO LA SALIDA":GOSUB 600
210 A=STICK(0):IF A=0 THEN 210
220 A=(A-1)/2:DR=(DR + A) MOD 4
230 IF M(XM+DX(DR)/2,YM+DY(DR)/2) THEN D
R=(DR+4-A) MOD 4:GOTO 210
240 XM=XM+DX(DR):YM=YM+DY(DR)
250 GOTO 190
260 *SLBRUTINA

```

```

270 R0=INT(RND(1)*4):R=R2
280 IF M(X+DX(R),Y+DY(R))=0 GOTO 300
290 R=(R+1)MOD 4:IF R<>R0 GOTO 280 ELSE
CN=0:RETURN
300 M(X+DX(R)/2,Y+DY(R)/2)=1:X=X+DX(R):Y
=Y+DY(R)
310 M(X,Y)=1:C=C-1:CN=-1:RETURN
320 CLS
330 X=XM:Y=YM:X0=128:Y0=92:LX=128:LY=92
340 U0=DX(DR)/2:U0=DY(DR)/2
350 U1=DX((DR+1) MOD 4)/2:U1=DY((DR+1) M
OD 4)/2:U2=DX((DR+3) MOD 4)/2:U2=DY((DR+
3) MOD 4)/2
360 IF M(X,Y) THEN IF FNE(X-U0,Y-U0) GO
TO 450 ELSE 460
370 IF M(X+U1,Y+U1) THEN GOSUB 430 ELSE
GOSUB 400
380 LX=-LX:IF M(X+U2,Y+U2) THEN GOSUB 43
0 ELSE GOSUB 400
390 LX=ABS(LX*.8):LY=LY*.8:X=X+U0:Y=Y+U0
:GOTO 360
400 LINE(X0+LX,Y0+LY)-(X0+LX,Y0+LY)
410 LINE(X0+LX,Y0+LY*.8)-(X0+LX*.8,Y0+LY
*.8)
420 LINE-(X0+LX*.8,Y0-LY*.8):LINE -(X0+L
X,Y0-LY*.8):RETURN
430 LINE(X0+LX,Y0+LY)-(X0+LX*.8,Y0+LY*.8
)
440 LINE(X0+LX,Y0-LY)-(X0+LX*.8,Y0-LY*.8
):RETURN
450 LINE(X0-LX,Y0-LY)-(X0+LX,Y0+LY),,BF:
RETURN
460 LINE(X0-LX,Y0-LY)-(X0+LX,Y0+LY),,B:R
ETURN
470 IF ERL=280 THEN IF ERR=5 OR ERR=9 TH
EN RESUME 290
480 ON ERROR GOTO 0
490 END
500 FOR I=1 TO 3:PLAY"T20005L62GECGECGEC
GECGECGEC":NEXT I:RETURN
600 LOCATE 2,15:PRINT"QUIERES CONTINUAR(
S/N):":A$=INKEY$:IF A$="S" THEN 5 ELSE I
F A$="N" THEN END ELSE IF A$<>"S" OR A$<>
"N" GOTO 600

```

## PROGRAMA PARA DIBUJAR CON LOS CURSORES

```
10 ON STOP GOSUB 250:STOP ON
20 CLS
30 COLOR 1,15,15
40 SCREEN 2,0
50 OPEN "GRP:"FOR OUTPUT AS #1
60 PRESET(17,1)
70 PRINT #1,"TECLAS DE COMANDO PARA DIBU
JAR"
80 PRESET(17,32)
90 PRINT#1,"F1:PALETA DE COLORES(selecci
on con cursor)"
100 PRESET(17,56)
110 PRINT#1,"F2:LINEAS (longitud y posic
ion final con cursor)"
120 PRESET(17,80)
130 PRINT#1,"F3: CIRCUNFERENCIAS(forma y
ta tamaño con cursor)"
140 PRESET(17,104)
150 PRINT#1,"F4:CUADRADOS COLOREADOS (fo
rma y tamaño con cursor)"
160 PRESET(17,128)
170 PRINT#1,"F5:COLOR A SUPERFICIES(solo
si estan cerradas)"
180 PRESET(17,152)
190 PRINT#1,"F6:BORRADO DEL DIBUJO"
200 PRESET(17,170)
210 PRINT#1,"PARA EJECUTAR COMANDOS PULS
E"
220 PRESET(17,183)
230 PRINT#1,"TECLA RETURN"
240 A$=INKEY$:IF A$=""THEN240
250 CLS
260 COLOR 1,15,15
270 SCREEN2
280 FOR D=0TO249STEP16
290 LINE(D,187)-(D+16,190),D/16,BF
300 NEXTD
310 SPRITE$(1)=CHR$(&H0)+CHR$(&H10)+CHR$
(&H10)+CHR$(&H7C)+CHR$(&H10)+CHR$(&H10)+
CHR$(&H0)-CHR$(&H0)
320 C=0
```

```

330 X=128:Y=96
340 PUT SPRITE 3,(X-3,Y-4),1,1
350 A$=INKEY$
360 PSET (X,Y),C
370 ON KEY GOSUB 650,500,780,940,620,109
0
380 KEY(1)ON:KEY(2)ON:KEY(3)ON:KEY(4)ON:
KEY(5)ON:KEY(6)ON
390 IF A$="" THEN 340
420 IF X=256 THEN X=X-1
410 IF Y=186 THEN Y=Y-1
420 IF X=0 THEN X=X+1
430 IF Y=0 THEN Y=Y+1
440 IF A$=CHR$(28) THEN X=X+1
450 IF A$=CHR$(29) THEN X=X-1
460 IF A$=CHR$(30) THEN Y=Y-1
470 IF A$=CHR$(31) THEN Y=Y+1
480 IF A$=CHR$(84) THEN 600
490 GOTO 340
500 A=X:B=Y
510 PSET(X,Y),C
520 A$=INKEY$
530 IF A$=CHR$(13)THEN RETURN
540 IF A$=""THEN520
550 LINE(A,B)-(X,Y),15
560 IF A$=CHR$(28)THENX=X+1
570 IF A$=CHR$(29)THENX=X-1
580 IF A$=CHR$(30)THENY=Y-1
590 IF A$=CHR$(31)THENY=Y+1
600 LINE(A,B)-(X,Y),C
610 GOTO 520
620 X=X:Y=Y
630 PAINT(X-2,Y+2),C
640 RETURN
650 N=0
660 LINE(N,186)-(N+16,192),1,B
670 C=POINT(N+5,190)
680 A$=INKEY$
690 IF A$="" THEN 680
700 LINE(N,186)-(N+16,192),15,B
710 IF A$=CHR$(13)THEN RETURN
720 IF A$=CHR$(28)THENN=N+16
730 IF A$=CHR$(29)THENN=N-16

```

```

740 IF N=-16 THEN N=0
750 IF N=256 THEN N=N-16
760 LINE(N,186)-(N+16,192),1,B
770 GOTO 660
780 F=X:G=Y
790 A$=INKEY$
800 A=ABS(F-X):B=ABS(G-Y):H=A+1:I=B+1:T=
ABS(I/H+.1)
810 CIRCLE(X,Y),A,C,...,T
820 IF A$="" THEN 790
830 IF A$=CHR$(13) THEN RETURN
840 CIRCLE(X,Y),A,15,...,T
850 IF A$=CHR$(28) THEN F=F+1
860 IF A$=CHR$(29) THEN F=F-1
870 IF A$=CHR$(30) THEN G=G-1
880 IF A$=CHR$(31) THEN G=G+1
890 IF F=0 THEN F=F+1
900 IF F=255 THEN F=F-1
910 IF G=0 THEN G=G+1
920 IF G=185 THEN G=G-1
930 GOTO 790
940 D=X:E=Y
950 A$=INKEY$
960 IF A$=CHR$(13) THEN RETURN
970 IF A$="" THEN 950
980 LINE(D,E)-(X,Y),15,BF
990 IF A$=CHR$(28) THEN X=X+1
1000 IF A$=CHR$(29) THEN X=X-1
1010 IF A$=CHR$(30) THEN Y=Y-1
1020 IF A$=CHR$(31) THEN Y=Y+1
1030 IF X=0 THEN X=X+1
1040 IF X=255 THEN X=X-1
1050 IF Y=0 THEN Y=Y+1
1060 IF Y=185 THEN Y=Y-1
1070 LINE(D,E)-(X,Y),C,BF
1080 GOTO 950
1090 GOTO 250

```

## ANEXO 5: MENSAJES Y CODIGOS DE ERROR

Código	Mensaje	Explicación
1	NEXT Without FOR	Existe un nombre de variable en una instrucción NEXT que no corresponde a la variable de la instrucción FOR o no existe el NEXT.
2	Syntax error	Una línea de programa contiene caracteres incorrectos (puntuación, parentesis no emparejados, error mecanográfico...)
3	RETURN without GOSUB	Se encontró una instrucción RETURN sin que se hubiera incorporado un comando GOSUB. GOSUB no emparejado.
4	Out of DATA	Se efectuó una instrucción READ, pero no quedaban datos por leer en la instrucción DATA.
5	Illegal function call	Un parámetro fuera de gama se ha asignado a una función matemática o de STRING. Un Illegal function call puede ocurrir también como resultado de. <ol style="list-style-type: none"><li>1. Una instrucción errónea o excesivamente larga.</li><li>2. Un argumento negativo o cero con LOG</li><li>3. Un argumento negativo con SQR</li><li>4. Un argumento impropio para MID \$, LEFT \$, RIGHT \$, INP, OUT, PEEK, POKE, TAB, SPC, STRIN \$, SPACE \$, INSTR \$ o ON... GOTO</li></ol>

- |    |                            |  |
|----|----------------------------|--|
| 6  | Overflow                   | El resultado de un cálculo es demasiado largo para ser representado en el formato BASIC.   |
| 7  | Out of memory              | Un programa es demasiado largo, tiene demasiadas instrucciones, demasiados GOSUB, demasiados FOR, demasiadas variables o expresiones demasiado complejas.  |
| 8  | Undefined line number      | Una línea referenciada en un GOTO, GOSUB, IF... THEN... ELSE es una línea inexistente.   |
| 9  | Subscript out of range     | Un elemento de la matriz está referenciado con una cifra fuera de las dimensiones de la matriz, o posee un número erróneo de subíndices.   |
| 10 | Redimensioned array        | Una matriz puede dimensionarse solo una vez, con DIM. Si se efectúan dos DIM para una misma matriz aparece este mensaje, si se usa una variable de matriz antes de que ésta sea dimensionada, se ejecuta una operación DIM automáticamente dimensionándose en 10 elementos. Si se pretende dimensionar posteriormente, se generará el mensaje de error citado. |
| 11 | División by zero           | Al ser una operación sin sentido matemático aparece el mensaje de error.   |
| 12 | Illegal direct             | Una instrucción que solamente puede incorporarse en un programa se ha introducido en Modo directo.   |
| 13 | Type mismatch              | Un número se ha asignado a una cadena o viceversa; una función que esperaba un número recibe una cadena o viceversa.   |
| 14 | Out of string space        | Unas variables string han sobrepasado la memoria disponible. El BASIC asignará dinámicamente espacio para cadenas hasta que se agoten los recursos de memoria.   |
| 15 | String too long            | Aparece este mensaje si se pretende abarcar en una cadena más de 255 caracteres.   |
| 16 | String formula too complex | La expresión de la cadena es demasiado larga o compleja. Debería dividirse al menos en dos partes para que pueda ser elaborada.  |

17	Can't continue.	El programa no puede continuar porque <ol style="list-style-type: none"> <li>1. Se ha producido un error.</li> <li>2. se ha modificado durante una interrupción de su ejecución.</li> <li>3. No existe</li> </ol>
18	Undefined user function	Se usó como referencia una función definida por el usuario, pero la misma no había sido definida usando la instrucción DEF FN.
19	Device I/O error	Error al conectar cassette, impresora, pantalla. Es un error <i>fatal</i> ya que el programa no puede recuperarse.
20	Verify error	El programa actual es distinto del grabado en cassette. (Grabación incorrecta)
21	NO RESUME	Se ha entrado una rutina de tratamiento de errores, pero no hay ninguna instrucción RESUME.
22	RESUME without error	Se encuentra una instrucción RESUME antes de que se haya entrado en una rutina de tratamiento de errores.
23	Unprintable error	No hay mensaje de error por la condición de error que se da. Usualmente es el caso de una instrucción ERROR con un código de error no definido.
24	Missing operand	Una expresión contiene un operador no seguido de operando.
25	Line buffer overflow	Se ha entrado una línea con demasiados caracteres.
26-49		Estos códigos no tienen definición. Se reservan para futuras expansiones del BASIC.
50	Field overflow	Una instrucción FIELD intentó colocar más bytes de los especificados para la longitud de registro de un fichero aleatorio.
51	Internal error	Se ha producido un funcionamiento defectuoso en el lenguaje BASIC.
52	Bad file number	En una instrucción o comando se cita un número de fichero que no se ha abierto anteriormente o que está fuera de la gama de número de fichero especificados en la instrucción MAXFILES

- |        |                          |   |
|--------|--------------------------|---|
| 53     | File not found           | En una instrucción LOAD, OPEN o KILL se cita un fichero que no existe en la memoria.  |
| 54     | File already open        | Se ha encontrado una instrucción de salida para un fichero que ya estaba abierto, o se ha entrado un comando KILL para un fichero que está abierto.   |
| 55     | Input past end           | Se ha ejecutado una instrucción INPUT tras haberse entrado todos los datos en el fichero o a partir de un fichero nulo. Para evitar este error usar la función EOF, para detectar el final del fichero. |
| 56     | Bad file name            | Se ha usado una forma ilegal de nombre de fichero p.e.: LOAD, SAVE, KILL, NAME, etc.  |
| 57     | Direct statement in file | Se encuentra una instrucción directa al cargar con LOAD un fichero en formato ASCII. La ejecución de LOAD se ha interrumpido.   |
| 58     | Sequential I/O only      | Una instrucción de acceso aleatorio es utilizada para un fichero secuencial.  |
| 59     | File not open...         | El fichero designado con PRINT #, INPUT #, etc. no se ha abierto.   |
| 60-255 |                          | Estos códigos no tienen definición. El usuario puede crear su propio código de errores dentro de esta secuencia.  |

