



secoinsa

FM-7

F-BASIC MANUAL DE REFERENCIA

CODIGO FM-7 102014-1-E

ORDENADOR PERSONAL

FM-7

F-BASIC MANUAL DE REFERENCIA

CODIGO COMERCIAL: 500088004

CODIGO DE PRODUCTO: 102014-1-E

VERSION 1 SEPTIEMBRE-85

3.12.6.	DSKI\$ (inicializar disco\$)	3-276
3.12.7.	DSKF (fichero de disco)	3-278
3.12.8.	INPUT\$ (entrada \$)	3-279
3.12.9.	INKEY\$ (tecla de entrada \$)	3-281

APENDICES

Apéndice A	Mapa de memoria	A-1
Apéndice B	Código de caracteres	A-5
Apéndice C	Mensajes de error del F-BASIC	A-6
Apéndice D	Instrucciones del F-BASIC	A-11

(c) 1983 by FUJITSU Ltd.

Reservados todos los derechos de reproducción, transmisión o reexportación en cualquier forma o por cualquier medio, electrónico o mecánico, incluyendo fotocopia, grabación en cinta o disco magnéticos, medio de almacenamiento de información o sistema de recuperación, tanto para la totalidad como para cualquier parte de este libro, sin permiso escrito del editor.

Depósito Legal: M-39359-1985

ISBN: Exento

Imprime: Bouncopy, S.A. - Avda. del Brasil, 7 - 28020 Madrid.

PREFACIO

Este manual describe la sintaxis del lenguaje de programación F-BASIC para el ordenador personal FUJITSU MICRO 7 (FM-7). El F-BASIC es una versión avanzada del BASIC de Microsoft, que le proporciona una gama muy variada de potentes funciones para facilitar la creación de programas de aplicación, al usar los recursos del FM-7.

Este manual ofrece una explicación detallada de cada instrucción del F-BASIC. Para consultar el método de operación del sistema, ver el manual ORDENADOR PERSONAL FM-7. MANUAL DEL USUARIO: DESCRIPCION DEL SISTEMA, código de producto 102044-1-E o el manual ORDENADOR PERSONAL FM-7. MANUAL DE INSTALACION Y USO, código de producto 102024-1-E.

Desproteccion de Programas

00 D1 - FF → 00

ó

01 E7 - KK → FF

01 E8 - KK → FF

Se sale con Break ó CTRL + C

con Break + Reset

Comentarios

- (1) Para más información acerca de este manual y otros existentes sobre el FM-7 consultar en la tienda o centro donde se adquirió.
- (2) Antes de conectar las unidades optativas y los periféricos a su FM-7, le rogamos que lea el Capítulo 3 del manual ORDENADOR PERSONAL FM-7. MANUAL DE INSTALACION Y USO, código de producto 102024-1-E.

INDICE

PAGINA

1.	F-BASIC	
1.1.	Descripción esquemática	1-1
1.2.	Versiones del F-BASIC	1-2
2.	REGLAS DE PROGRAMACION DEL F-BASIC	
2.1.	Formato de línea	2-1
2.2.	Referencia a los números de línea	2-2
2.3.	Conjunto de caracteres	2-2
2.4.	Constantes	2-4
2.4.1.	Constantes cadena	2-4
2.4.2.	Constantes numéricas	2-4
2.5.	Variables	2-7
2.5.1.	Nombres de variables y caracteres de declaración de tipo	2-8
2.5.2.	Matrices	2-10
2.5.3.	Palabras clave	2-12
2.5.4.	Palabras reservadas	2-13
2.6.	Conversión de tipo de datos	2-15
2.7.	Expresiones	2-19
2.7.1.	Expresiones aritméticas	2-19
2.7.2.	Expresiones relacionales	2-20
2.7.3.	Expresiones lógicas	2-22
2.7.4.	Expresiones cadena	2-25
2.7.5.	Prioridad de operadores	2-25
2.8.	Descriptor de fichero	2-26
2.9.	Inicialización	2-29
2.10.	Gestión y estructura de ficheros	2-32
2.10.1.	Grabador de datos	2-32
2.10.2.	Disquete flexible	2-34
3.	INSTRUCCIONES EN F-BASIC	
3.1.	Comandos	3-5
3.1.1.	AUTO (auto)	3-5
3.1.2.	DELETE (suprimir)	3-7
3.1.3.	LIST (listar)	3-8
3.1.4.	UNLIST (no listar)	3-11
3.1.5.	LLIST (listar por impresora)	3-12
3.1.6.	RENUM (renumerar)	3-14
3.1.7.	NEW (nuevo)	3-17
3.1.8.	CLEAR (borrar)	3-18
3.1.9.	CONT (continuar)	3-20
3.1.10.	RUN (procesar)	3-22

3.1.11.	LOAD (cargar)	3-24
3.1.12.	LOAD? (cargar)?	3-25
3.1.13.	SAVE (guardar)	3-26
3.1.14.	FILES (ficheros)	3-28
3.1.15.	NAME (nombrar)	3-30
3.1.16.	KILL (eliminar fichero del disquete)	3-31
3.1.17.	MERGE (mezclar)	3-32
3.1.18.	SKIPF (saltar fichero)	3-34
3.1.19.	DSKINI (inicializar disco)	3-35
3.1.20.	EXEC (ejecutar)	3-36
3.1.21.	LOADM (cargar programa máquina)	3-37
3.1.22.	SAVEM (guardar programa máquina)	3-39
3.1.23.	HARDC (copia impresa)	3-41
3.1.24.	MON (monitor)	3-43
3.1.25.	TERM (terminal)	3-48
3.1.26.	EDIT(editar)	3-51
3.2.	Sentencias generales	3-53
3.2.1.	DEF FN (definir funciones)	3-53
3.2.2.	DEF USR (definir usuario)	3-55
3.2.3.	DEFINIT/SNG/DBL/STR (definir simple/doble/ cadena)	3-59
3.2.4.	REM (comentario)	3-61
3.2.5.	END (fin)	3-62
3.2.6.	FOR NEXT (para ... siguiente)	3-63
3.2.7.	NEXT (siguiente)	3-65
3.2.8.	GOTO (ir a)	3-66
3.2.9.	ON GOTO (en ... ir a)	3-67
3.2.10.	GOSUB (ir a subrutina)	3-69
3.2.11.	RETURN (retorno)	3-70
3.2.12.	ON GOSUB (en ... ir a subrutina)	3-71
3.2.13.	STOP (parar)	3-73
3.2.14.	IF THEN ELSE (si ...,entonces ...)	3-75
3.2.15.	WHILE WEND (mientras)	3-77
3.2.16.	WEND (fin de mientras)	3-79
3.2.17.	LET (hacer)	3-80
3.2.18.	SWAP (intercambiar)	3-81
3.2.19.	DIM (dimensión)	3-82
3.2.20.	POKE (poner)	3-84
3.2.21.	DATA (datos)	3-85
3.2.22.	READ (leer)	3-87
3.2.23.	RESTORE (restaurar)	3-88
3.2.24.	LSET, RSET (ajustar a la izquierda, idem derecha)	3-90
3.2.25.	RANDOMIZE (aleatorizar)	3-92
3.2.26.	ERROR (error)	3-94

3.2.27.	ON ERROR GOTO (en error ir a)	3-96
3.2.28.	RESUME (reasumir)	3-97
3.2.29.	BEEP (señal acustica)	3-98
3.2.30.	MOTOR (motor)	3-99
3.2.31.	TRON (activar traza)	3-100
3.2.32.	TROFF (desactivar traza)	3-102
3.2.33.	CHAIN (encadenar)	3-103
3.2.34.	COMMON (común)	3-106
3.2.35.	ERASE (borrar)	3-108
3.3.	Sentencias de entrada/salida	3-109
3.3.1.	INPUT (entrada)	3-109
3.3.2.	LINE INPUT (entrada de línea)	3-111
3.3.3.	PRINT (imprimir)	3-113
3.3.4.	LPRINT (imprimir por impresora)	3-115
3.3.5.	PRINT USING (imprimir con formato)	3-116
3.3.6.	LPRINT USING (idem por impresora)	3-121
3.3.7.	OPEN (abrir)	3-122
3.3.8.	CLOSE (cerrar)	3-125
3.3.9.	INPU	
3.3.4.	LPRINT (imprimir por impresora)	3-115
3.3.5.	PRINT USING (imprimir con formato)	3-116
3.3.6.	LPRINT USING (idem por impresora)	3-121
3.3.7.	OPEN (abrir)	3-122
3.3.8.	CLOSE (cerrar)	3-125
3.3.9.	INPUT# (entrada por fichero #)	3-126
3.3.10.	PRINT# (imprimir por fichero #)	3-127
3.3.11.	LINE INPUT# (entrada de línea idem#)	3-129
3.3.12.	FIELD (campo)	3-131
3.3.13.	GET (obtener)	3-133
3.3.14.	PUT (poner)	3-135
3.3.15.	DSKO\$ (salida a disco \$)	3-136
3.4.	Control de pantalla y funciones gráficas	3-137
3.4.1.	WIDTH (ancho)	3-137
3.4.2.	CONSOLE (consola)	3-138
3.4.3.	COLOR (color)	3-140
3.4.4.	SCREEN (pantalla)	3-145
3.4.5.	CLS (borrar pantalla)	3-148
3.4.6.	LOCATE (colocar)	3-149
3.4.7.	PSET (establecer punto)	3-151
3.4.8.	PRESET (borrar punto)	3-153
3.4.9.	LINE (línea)	3-154
3.4.10.	CONNECT (conectar)	3-158
3.4.11.	SYMBOL (símbolo)	3-160
3.4.12.	GET@ (obtener @)	3-162
3.4.13.	PUT@ (poner @)	3-170

3.4.14.	CIRCLE (círculo)	3-174
3.4.15.	GCURSOR (cursor gráfico)	3-177
3.4.16.	PAINT (pintar)	3-179
3.5.	Funciones musicales	3-181
3.5.1.	PLAY (componer)	3-181
3.5.2.	SOUND (sonido)	3-182
3.6.	Funciones de teclas de función programables	3-194
3.6.1.	KEY (tecla)	3-194
3.6.2.	KEY LIST (lista de teclas)	3-195
3.6.3.	KEY(n) ON/OFF/STOP (tecla (n) activar/ desactivar/parar)	3-197
3.6.4.	ON KEY(n) GOSUB (en tecla (n) ir a subrutina)	3-199
3.7.	Funciones de interrupción del reloj	3-200
3.7.1.	ON TIME GOSUB (en hora ir a subrutina) ..	3-200
3.7.2.	TIME (hora)	3-202
3.7.3.	TIME ON/OFF/STOP (hora activar/desactivar/ aplazar)	3-203
3.7.4.	ON INTERVAL GOSUB (en intervalo ir a subrutina)	3-204
3.7.5.	INTERVAL (intervalo)	3-206
3.7.6.	INTERVAL ON/OFF/STOP (intervalo activado/ desactivado/parar)	3-207
3.8.	Funciones de control de línea	3-208
3.8.1.	ON COM(n) GOSUB (en comunicación ir a subrutina)	3-208
3.8.2.	COM(n) ON/OFF/STOP (comunicación(n) activada/desactivada/parar)	3-210
3.8.3.	OPEN (abrir)	3-211
3.8.4.	CLOSE (cerrar)	3-213
3.8.5.	INPUT# (entrada canal#)	3-214
3.8.6.	LINE INPUT# (entrada de línea canal#)	3-215
3.8.7.	PRINT# (impresión canal#)	3-216
3.8.8.	LIST (listar)	3-217
3.9.	Funciones aritméticas	3-218
3.9.1.	ABS (absoluto)	3-218
3.9.2.	ATN (arcotangente)	3-219
3.9.3.	COS (coseno)	3-220
3.9.4.	EXP (exponencial)	3-221
3.9.5.	FIX (truncar a entero)	3-222
3.9.6.	INT (parte entera)	3-223
3.9.7.	LOG (logaritmo)	3-224
3.9.8.	RND (aleatorio)	3-225
3.9.9.	SGN (signo)	3-227

3.9.10.	SIN (seno)	3-228
3.9.11.	SQR (raíz cuadrada)	3-229
3.9.12.	TAN (tangente)	3-230
3.9.13.	CSNG (convertir a simple)	3-231
3.9.14.	CDBL (convertir a doble)	3-232
3.9.15.	CINT (convertir a número entero)	3-233
3.10.	Funciones de cadena	3-234
3.10.1.	CHR\$ (carácter \$)	3-234
3.10.2.	HEX \$ (hexadecimal \$)	3-235
3.10.3.	LEFT\$ (izquierda \$)	3-236
3.10.4.	MID\$ (interior \$)	3-237
3.10.5.	OCT\$ (octal \$)	3-239
3.10.6.	RIGHT\$ (derecha \$)	3-240
3.10.7.	SPACE\$ (espacio \$)	3-241
3.10.8.	STR\$ (cadena \$)	3-242
3.10.9.	STRING\$ (cadena \$)	3-243
3.10.10.	ASC (ASCII)	3-244
3.10.11.	INSTR (en cadena)	3-245
3.10.12.	LEN (longitud)	3-247
3.10.13.	VAL (valor)	3-248
3.11.	Funciones generales	3-250
3.11.1.	CSRLIN (línea del cursor)	3-250
3.11.2.	POS (posición)	3-251
3.11.3.	LPOS (posición de línea)	3-252
3.11.4.	POINT (punto)	3-253
3.11.5.	ERR/ERL (número de error, línea de error) ..	3-254
3.11.6.	VARPTR (puntero de variable)	3-255
3.11.7.	USR (usuario)	3-257
3.11.8.	PEEK (mirar)	3-258
3.11.9.	FRE (libre)	3-259
3.11.10.	TIME\$ (hora \$)	3-260
3.11.11.	DATE \$ (fecha \$)	3-263
3.11.12.	SPC (espacio)	3-265
3.11.13.	TAB (tabulador)	3-266
3.11.14.	SCREEN (pantalla)	3-267
3.12.	Funciones de entrada/salida	3-269
3.12.1.	CVI/CSV/CVD (convertir a entero/simple/ doble)	3-269
3.12.2.	MKI\$/MKS\$/MKD\$ (hacer entero\$/simple\$/ doble\$)	3-271
3.12.3.	EOF (fin de fichero)	3-272
3.12.4.	LOF (longitud de fichero)	3-274
3.12.5.	LOC (contador de posición del fichero)	3-275

1. F-BASIC

1.1. Descripción esquemática

El BASIC (Beginner's All Purpose Symbolic Instruction Code, es decir, Código de Instrucciones Simbólico de Propósito General para Principiantes) se ha convertido en el lenguaje estándar de facto para ordenadores personales. Originalmente diseñado para uso en sistemas de tiempo compartido, el BASIC fue revisado para microordenadores y ordenadores personales por Microsoft Corporation en los Estados Unidos. El FUJITSU MICRO (FM-7) opera con el intérprete F-BASIC instalado en su unidad principal. El F-BASIC es el BASIC estándar, con algunas ampliaciones hechas a medida para las posibilidades electrónicas del microordenador Fujitsu, FM-7.

El F-BASIC está disponible en tres versiones: V1.0, V2.0 y V3.0. El FM-7 usa la versión V3.0. Este manual describe las instrucciones de las tres versiones.

Para asignar un valor a una variable, se usa una sentencia de asignación INPUT o READ. En cualquier sentencia, el tipo de variable (cadena o numérica) debe ser el mismo que el tipo de constante asignado a la variable.

Cada variable numérica se inicializa con cero hasta que se le asigne un valor. Cada variable cadena se inicializa con una cadena vacía hasta que se le asigne un carácter.

2.5.1. Nombres de variables y caracteres de declaración de tipo

Un nombre de variable BASIC puede estar formado por cualquier número de caracteres que quepa en una línea, pero el BASIC solamente utilizará los 16 primeros. Para formar un nombre de variable pueden utilizarse caracteres alfabéticos y numéricos, pero debe comenzar por un carácter alfabético. El BASIC no distingue entre mayúsculas y minúsculas. Si un nombre de variable se introduce usando minúsculas, entonces se convierte automáticamente a mayúsculas, antes de almacenarse en memoria.

Un nombre de variable no puede ser una palabra reservada, ni comenzar con una palabra reservada.

(Ejemplo)

- | | |
|-------|---|
| LIST | Es una palabra reservada, y no se puede usar como nombre de variable. |
| LISTA | Esta cadena de caracteres comienza con la palabra reservada LIST, y no se puede usar como nombre de variable. |
| ALIST | Esta cadena de caracteres sí puede usarse como nombre de variable. |

Las palabras reservadas son palabras claves del BASIC (nombres de comandos, nombres de instrucciones, nombres de funciones y nombres de operadores. Para más detalles ver la Sección 2.5.4.).

Una variable puede representar tanto un número como una cadena.

1.2. Versiones del F-BASIC

A continuación se resumen las tres versiones del F-BASIC.

Versiones del F-BASIC

Versión	Medio	Nombre	Abrev.	Modelo FM
V1.0	ROM en el FM-8	modo ROM F-BASIC V1.0	V1.0/ ROM	FM-8
	ROM en el FM-8 y disco del sistema (suministrado con la unidad de disco)	modo DISCO F-BASIC V1.0	V1.0/ DISCO	
V2.0	Mini disquete flexible	versión 5" F-BASIC V2.0	V2.0/5	FM-8
	Disco flexible estándar	versión 8" F-BASIC V2.0	V2.0/8	
V3.0	ROM en el FM-7	modo ROM F-BASIC V3.0	V3.0/ ROM	FM-7
	ROM en el FM-7 y disco del sistema V3.0	modo DISCO F-BASIC V3.0	V3.0/ DISCO	

P2

(1) F-BASIC V1.0

La versión V1.0 del F-BASIC opera en ROM o en modo DISCO.

(2) F-BASIC V2.0

El F-BASIC V2.0 es igual que el F-BASIC V1.0, más las siguientes prestaciones:

Sentencias CHAIN, COMMON, ERASE, LLIST, LPRINT y LPRINT USING, y la función LPOS.

-
- El comando /sentencia TERM con un procedimiento de recuperación de errores mejorado.
 - La sentencia PRINT USING con caracteres de control de formato adicionales.
 - La sentencia OPEN con opciones adicionales de impresora.
 - Arranque automático de programas de usuario.

(3) F-BASIC V3.0

El F-BASIC V3.0 es una versión revisada del F-BASIC V2.0 en la que se han añadido unas prestaciones y se han eliminado otras.

Prestaciones Añadidas

- La sentencia SCREEN para activar y visualizar en pantalla las VRAMs.
- La sentencia COLOR para especificar códigos de paleta.
- Las sentencias PLAY y SOUND para producir sonidos musicales.

Prestaciones Eliminadas

- La sentencia ANPORT para interfaz con entradas analógicas.

El comando `PRINT` muestra los resultados de
las operaciones de cálculo.

La sentencia `PRINT USING` con caracteres de control de
formato específicos.

La sentencia `OPEN` con opciones adicionales de apertura.

Atajos automáticos de programas de usuario.

3. F-BASIC V3.0

El F-BASIC V3.0 es una versión mejorada del F-BASIC V2.0 en
la que se han añadido más prestaciones y se han eliminado
algunas.

Prestaciones Añadidas

La sentencia `SCREEN` para activar y desactivar el pantalla en
modo texto.

La sentencia `COLOR` para especificar códigos de colores.

La sentencia `PLAY SOUND` para producir sonidos musicales.

Prestaciones Eliminadas

La sentencia `IMPORT` para importar con comandos analógicos.

2. REGLAS DE PROGRAMACION DEL F-BASIC

2.1. Formato de línea

Un programa en BASIC consiste en un conjunto de líneas.

El formato de una línea es:

`nnnnn sentencia BASIC [:sentencia BASIC...]['comentario]`

nnnnn es el número de línea. Debe ser un número decimal de hasta 5 dígitos.

Una sentencia BASIC puede ser ejecutable o no ejecutable. Una sentencia ejecutable le dice al intérprete de BASIC que debe hacer algo durante la ejecución. En cambio, cuando el intérprete llega a una sentencia no ejecutable, simplemente pasa a la siguiente sentencia sin hacer nada. La sentencia PRINT es un ejemplo de sentencia ejecutable, mientras que DATA y REM son sentencias no ejecutables.

Se puede introducir más de una sentencia BASIC por línea. Cada sentencia en la misma línea ha de ir separada de la anterior mediante dos puntos(:).

Se puede incluir un comentario al final de cada línea, separándolo de la sentencia previa con una comilla simple ('). Una línea puede contener hasta 255 caracteres.

Una línea termina al pulsar la tecla RETURN.

(Ejemplo)

```
10 REM ***programa ejemplo***  
20 FOR I=1 TO 100:J=I :PRINT J:NEXT  
30 END'FIN del programa ejemplo
```


2.2. Referencia a los números de línea

Todas las líneas de un programa BASIC comienzan con un número de línea. Los números de línea indican la secuencia en la cual las sentencias se almacenan en memoria, y la secuencia en la que se ejecutan. La ejecución se lleva a cabo en orden ascendente de los números de línea.

Un número de línea puede ser especificado en sentencias tales como GOTO y GOSUB, y en comandos como LIST, DELETE y EDIT. El número de línea debe ser un entero entre 0 y 63999. Se puede introducir un punto (.) en un comando como LIST, AUTO, DELETE o EDIT para indicar la línea en uso. El punto puede usarse durante la edición o después de producirse un error.

(Ejemplo)

```
LIST.  
AUTO.  
DELETE. -100  
EDIT.
```

2.3. Conjunto de caracteres

El conjunto de caracteres del lenguaje F-BASIC contiene los siguientes caracteres:

Caracteres alfabéticos: 26 caracteres desde la A hasta la Z, tanto mayúsculas como minúsculas.

Caracteres numéricos: 10 caracteres del 0 al 9.

Caracteres griegos: Caracteres griegos.

Caracteres especiales: Símbolos especiales y caracteres gráficos.

Los siguientes símbolos especiales tienen un significado especial en BASIC.

Carácter	Nombre
	Blanco
=	Igual o símbolo de asignación
+	Signo más o el operador suma
-	Signo menos o el guión
*	El asterisco o el operador de multiplicación
/	La barra o el operador de división
\	La barra invertida o el operador de división entera
^	La flecha o el operador de exponenciación
(	Abrir paréntesis
)	Cerrar paréntesis
%	Tanto por ciento
#	Signo número
\$	Signo dólar
!	Punto de admiración
&	Ampersand
@	Marca en (at)
,	Coma
.	Punto
'	Comilla simple
;	Punto y coma
:	Dos puntos
?	Punto de interrogación
<	Signo menor que
>	Signo mayor que
"	Comillas dobles
_	Subrayado

2.4. Constantes

Una constante indica su propio valor. Cualquier cadena o número válido en F-BASIC puede ser una constante.

2.4.1. Constantes cadena

Una constante cadena es una serie de uno o más caracteres del repertorio de caracteres del F-BASIC, delimitada por comillas dobles. Una constante cadena puede tener una longitud entre 0 y 255 caracteres. Una cadena sin caracteres se llama cadena vacía.

(Ejemplo)

“GRACIAS”

“Nombre del artículo Precio unitario Ventas”

“” (cadena vacía)

2.4.2. Constantes numéricas

Una constante numérica puede ser tanto positiva como negativa. El BASIC no permite el uso de comas entre los dígitos de una constante numérica. Las constantes numéricas se clasifican en los cinco tipos siguientes:

(1) Tipo entero

Una serie de uno o más dígitos, precedida por un signo más (+) o menos (-) es una constante de tipo entero. Un número con un signo más o sin ningún signo se llama entero positivo. Un número con un signo menos se llama entero negativo. Un entero puede estar comprendido entre -32768 y +32767.

(Ejemplos)

1
+ 123
- 32767

(2) Tipo punto fijo

Una constante de punto fijo está compuesta de un signo, un entero, un punto decimal, y una fracción. El signo más (+) y cualquiera de las partes entera o fraccionaria (no ambas) se puede omitir. El punto decimal se representa con un punto.

(Ejemplos)

1.0
-123.11
.999

(3) Tipo punto flotante

Este tipo de constante es un número positivo o negativo representado en formato exponencial. Está compuesta de un signo, un entero, un punto decimal, una fracción y un exponente. El signo más y cualquiera de las partes, entera o fraccionaria (no ambas), o el punto decimal y la fracción se pueden omitir. El exponente se representa con el formato E[(+)]nn para simple precisión, y en el formato D[(+)]nn para doble precisión, siendo nn un entero sin signo.

(Ejemplos)

520.18E + 7
108E-18
0.1256E + 12
0.999999999D-8

(4) Tipo hexadecimal

Este tipo de constante está compuesta de dígitos hexadecimales (0 a 9 y A a F) con el prefijo &H. Puede tener una longitud entre 1 y 4 dígitos, (excluyendo los ceros a la izquierda), es decir, de &H0 a &HFFFF. Una entrada de un número hexadecimal en este formato se convierte, antes de la salida, en un número decimal sin signo.

(Ejemplos)

&H64
&H003F

(5) Tipo octal

Este tipo de constante se forma mediante dígitos octales (0 a 7), precedidos del prefijo &0 o &. Puede tener una longitud entre 1 y 6 dígitos (excluyendo los ceros a la izquierda), es decir desde &00 hasta &0177777. Una entrada de un número octal en este formato se convierte antes de la salida en un número decimal sin signo.

&00123
&0123456

Constantes numéricas con simple y doble precisión.

Las constantes numéricas se clasifican en simple precisión y doble precisión. Las constantes de simple precisión se almacenan con hasta 7 dígitos significativos y se imprimen con un máximo de 6. Una constante de doble precisión se almacena con hasta 17 dígitos significativos y se imprime con hasta 16 dígitos significativos. Los valores absolutos de una constante de simple o doble precisión están comprendidas entre aproximadamente $3,0 \times 10^{-39}$ y $1,7 \times 10^{38}$, con los números menores que $3,0 \times 10^{-39}$ interpretados como cero.

Las constantes numéricas de simple precisión pueden ser de los siguientes tipos:

(1) Una constante con 7 o menos dígitos significativos

(2) Una constante representada en formato exponencial con E

(3) Una constante que termina con el punto de admiración (!)

(Ejemplos)

13.6

-1.23E+ 6

862.1!

520

Las constantes numéricas de doble precisión pueden ser de los siguientes tipos:

(1) Una constante con más de 7 pero menos de 17 dígitos significativos.

(2) Una constante representada en formato exponencial con D.

(3) Una constante que termina con el signo (#)

(Ejemplos)

1368792702

-1.5670D-12

562.983#

852.138269012

2.5. Variables

Una variable es un nombre que se usa para representar un valor procesado en un programa BASIC. Al igual que una constante, una variable puede ser de tipo numérico o cadena. Una variable numérica representa siempre un valor numérico, mientras que una variable cadena representa una cadena de caracteres. La longitud de una variable cadena no es fija; se determina cuando se asigna su valor. Una variable cadena puede tener una longitud entre 0 y 255 caracteres.

Un nombre de variable puede estar seguido por un carácter de declaración de tipo indicando el tipo de valor que representa. El carácter de declaración de tipo permite que el mismo nombre de variable se use por diferentes variables y que representen diferentes tipos de variables. Los caracteres de declaración de tipo son los siguientes:

% Identifica una variable entera y requiere 2 bytes de memoria.

! Identifica una variable de simple precisión y requiere 4 bytes de memoria.

Identifica una variable de doble precisión y requiere 8 bytes de memoria.

\$ Identifica una variable cadena de hasta 255 caracteres.

Las instrucciones DEFINT, DEFSNG, DEFDBL y DEFSTR se usan también para declarar tipos de variables. Ver la Sección 3.2.3. para detalles. Un nombre de variable que no tenga ni sentencia de declaración de tipo ni carácter de declaración de tipo se considera que representa un número de simple precisión.

(Ejemplo 1)

A% Variable entera

MAX# Variable de doble precisión

L\$ Variable cadena

AB Variable de simple precisión

(Ejemplo 2)

VARIABLE%

VARIABLE!

VARIABLE#

VARIABLE\$

Todos estos nombres de variables son diferentes. Sin embargo, VARIABLE! y VARIABLE representan ambos el mismo número.

2.5.2. Matrices

Una matriz es una colección de elementos del mismo tipo. Una matriz tiene un nombre único, y sus elementos se identifican mediante subíndices. La sentencia DIM se usa para declarar una variable como matriz.

DIM nombre de variable (máximo valor del subíndice
[,máximo valor del subíndice]...)

El valor de un subíndice ha de estar comprendido entre 0 y 32766. No obstante, el valor máximo de un subíndice y el número de dimensiones de una matriz dependen del tamaño de memoria disponible. Cuando el valor máximo de un subíndice no sobrepasa de 10 en cada dimensión, no es necesario utilizar la instrucción DIM.

Para hacer referencia a un elemento de una matriz, hay que especificar el nombre de la variable, y a continuación entre paréntesis, un subíndice para cada dimensión. El formato es el siguiente:

nombre de variable(subíndice primera dimensión[,subíndice
segunda dimensión]...)

El número de subíndices entre paréntesis ha de coincidir con el número de dimensiones de la matriz. Cada subíndice debe ser una expresión numérica con un valor entre 0 y el valor máximo de la dimensión. Si el subíndice no es un entero, es redondeado.

(Ejemplo 1) Matriz de una dimensión

DIM A\$(6)

A\$(0)
A\$(1)
A\$(2)
A\$(3)
A\$(4)
A\$(5)
A\$(6)

(Ejemplo 2) Matriz de dos dimensiones

DIM B(2,3)

La matriz bidimensional puede considerarse como una tabla formada por filas y columnas, como se muestra a continuación:

		Fila			
Columna		B(0,0)	B(0,1)	B(0,2)	B(0,3)
		B(1,0)	B(1,1)	B(1,2)	B(1,3)
		B(2,0)	B(2,1)	B(2,2)	B(2,3)

2.5.3. Palabras clave

Las palabras clave tienen un significado y un uso predeterminado en BASIC. Todas las palabras claves son palabras reservadas, por lo que no se pueden utilizar como nombres de variables. Un nombre de variable tampoco puede comenzar con una palabra reservada. Las palabras clave en el F-BASIC son nombres de comandos, nombres de instrucciones y nombres de operadores.

Forma abreviada de las palabras clave

Las palabras clave que se detallan a continuación pueden codificarse en forma abreviada. En un listado de un programa, las palabras clave se imprimen con todas las letras aunque hayan sido introducidas en forma abreviada.

Palabras clave

CONT
LOAD
SAVE
SKIPF
MERGE
GOTO
GOSUB
RETURN
RANDOMIZE
LOCATE
COLOR
MOTOR
RUN
LIST
CONSOLE
WIDTH

Forma abreviada

C.
LO.
SA.
SK.
ME.
GO.
GOS.
RET.
RNDM.
LOC.
COL.
M.
R.
L.
CONS.
W.

2.5.4. Palabras reservadas

Clave: o - palabra reservada
x - palabra no reservada

Palabras reservadas	V1.0		V2.0	V3.0		Palabras reservadas	V1.0		V2.0	V3.0	
	R	D		R	D		R	D		R	D
ABS	o	o	o	o	o	CDBL	o	o	o	o	o
AND	o	o	o	o	o	CHAIN	x	x	o	o	o
ANPORT	o	o	o	o	o	CHR\$	o	o	o	o	o
ASC	o	o	o	o	o	CINT	o	o	o	o	o
ATN	o	o	o	o	o	CIRCLE	o	o	o	o	o
AUTO	o	o	o	o	o	CLEAR	o	o	o	o	o
						CLOSE	o	o	o	o	o
BEEP	o	o	o	o	o	CLS	o	o	o	o	o
COLOR	o	o	o	o	o	GO	o	o	o	o	o
COM	o	o	o	o	o						
COMMON	x	x	o	o	o	HARDC	o	o	o	o	o
CONNECT	o	o	o	o	o	HEX\$	o	o	o	o	o
CONSOLE	o	o	o	o	o						
CONT	o	o	o	o	o	IF	o	o	o	o	o
COS	o	o	o	o	o	IMP	o	o	o	o	o
CSNG	o	o	o	o	o	INKEY\$	o	o	o	o	o
CSRLIN	o	o	o	o	o	INPUT	o	o	o	o	o
CVD	x	o	o	x	o	INSTR	o	o	o	o	o
CVI	x	o	o	x	o	INT	o	o	o	o	o
CVS	x	o	o	x	o	INTERVAL	o	o	o	o	o
DATA	o	o	o	o	o	KEY	o	o	o	o	o
DATE	o	o	o	o	o	KILL	o	o	o	o	o
DEF	o	o	o	o	o						
DEFDBL	o	o	o	o	o	LEFT\$	o	o	o	o	o
DEFINT	o	o	o	o	o	LEN	o	o	o	o	o
DEFSNG	o	o	o	o	o	LET	o	o	o	o	o
DEFSTR	o	o	o	o	o	LINE	o	o	o	o	o
DELETE	o	o	o	o	o	LIST	o	o	o	o	o
DIM	o	o	o	o	o	LLIST	x	x	o	o	o
DSKF	x	o	o	x	o	LOAD	o	o	o	o	o
DSKI\$	x	o	o	x	o	LOC	x	o	o	x	o
DSKINI	x	o	o	x	o	LOCATE	o	o	o	o	o
DSKO\$	x	o	o	x	o	LOF	o	o	o	o	o
						LOG	o	o	o	o	o
EDIT	o	o	o	o	o	LPOS	x	x	o	o	o
ELSE	o	o	o	o	o	LPRINT	x	x	o	o	o
END	o	o	o	o	o	LSET	x	o	o	x	o
EOF	o	o	o	o	o						

Palabras reservadas	V1.0		V2.0	V3.0		Palabras reservadas	V1.0		V2.0	V3.0	
	R	D		R	D		R	D		R	D
EQV	o	o	o	o	o	MERGE	o	o	o	o	o
ERASE	x	x	o	o	o	MID\$	o	o	o	o	o
ERL	o	o	o	o	o	MKD\$	x	o	o	x	o
ERR	o	o	o	o	o	MKI\$	x	o	o	x	o
ERROR	o	o	o	o	o	MKS\$	x	o	o	x	o
EXEC	o	o	o	o	o	MOD	o	o	o	o	o
EXP	o	o	o	o	o	MON	o	o	o	o	o
						MOTOR	o	o	o	o	o
FIELD	x	o	o	x	o						
FILES	o	o	o	o	o	NAME	x	o	o	x	o
FIX	o	o	o	o	o	NEW	o	o	o	o	o
FN	o	o	o	o	o	NEXT	o	o	o	o	o
FOR	o	o	o	o	o	NOT	o	o	o	o	o
FRE	o	o	o	o	o						
						OCT\$	o	o	o	o	o
GCURSOR	o	o	o	o	o	OFF	o	o	o	o	o
GET	o	o	o	o	o	ON	o	o	o	o	o
OPEN	o	o	o	o	o	SPACE\$	o	o	o	o	o
OR	o	o	o	o	o	SPC	o	o	o	o	o
						SQR	o	o	o	o	o
PAINT	o	o	o	o	o	STEP	o	o	o	o	o
PEEK	o	o	o	o	o	STOP	o	o	o	o	o
PEN	o	o	o	o	o	STR\$	o	o	o	o	o
PLAY	x	x	x	o	o	STRING\$	o	o	o	o	o
POINT	o	o	o	o	o	SUB	o	o	o	o	o
POKE	o	o	o	o	o	SWAP	o	o	o	o	o
POS	o	o	o	o	o	SYMBOL	o	o	o	o	o
PRESET	o	o	o	o	o						
PRINT	o	o	o	o	o	TAB	o	o	o	o	o
PSET	o	o	o	o	o	TAN	o	o	o	o	o
PUT	o	o	o	o	o	TERM	o	o	o	o	o
						THEN	o	o	o	o	o
RANDOMIZE	o	o	o	o	o	TIME	o	o	o	o	o
READ	o	o	o	o	o	TO	o	o	o	o	o
REM	o	o	o	o	o	TROFF	o	o	o	o	o
RENUM	o	o	o	o	o	TRON	o	o	o	o	o
RESTORE	o	o	o	o	o						
RESUME	o	o	o	o	o	UNLIST	o	o	o	o	o
RETURN	o	o	o	o	o	USING	o	o	o	o	o
RIGHT\$	o	o	o	o	o	USR	o	o	o	o	o
RND	o	o	o	o	o						
RSET	o	o	o	o	o	VAL	o	o	o	o	o
RUN	o	o	o	o	o	VARPTR	o	o	o	o	o
SAVE	o	o	o	o	o	WEND	o	o	o	o	o
SCREEN	o	o	o	o	o	WHILE	o	o	o	o	o
SGN	o	o	o	o	o	WIDTH	o	o	o	o	o
SIN	o	o	o	o	o						
SKIPF	o	o	o	o	o	XOR	o	o	o	o	o
SOUND	x	x	x	o	o						

Clave: R = modo ROM

D = modo de disco

2.6. Conversión de tipo de datos

Los datos numéricos en una sentencia BASIC se convierten, durante la ejecución, de un tipo a otro según lo requerido. En la conversión se aplican las siguientes reglas:

- (1) Cuando los datos numéricos de un tipo dado se asignan a una variable numérica de otro tipo, los datos numéricos se convierten al tipo declarado para el nombre de la variable.

(Ejemplos)

```
10 A%=12.34
20 PRINT A%
```

```
RUN
12
Ready
```

- (2) Cuando los datos numéricos de un tipo dado se asignan a una variable numérica de menor precisión, se redondean añadiendo 1 al dígito menos significativo de la variable si el más significativo de los dígitos eliminados es mayor o igual que 5. En el ejemplo siguiente, los datos numéricos de doble precisión se asignan a una variable de simple precisión.

(Ejemplos)

```
10 A=12.3456789#
20 PRINT A
```

```
RUN
12.3457
Ready
```

Cuando se asignan datos reales a una variable entera, el número se redondea al entero más cercano, contando como 1 las fracciones de 0,5 y mayores, y contando como 0 las fracciones menores de 0,5. Si los datos convertidos sobrepasan el valor máximo permitido para los datos enteros, entonces se comunica un error.

(Ejemplo 1)

```
10 A%=12.34
20 B%=35.56
30 PRINT A%, B%
```

```
RUN
12          36
Ready
```

(Ejemplo 2)

```
10 A%=1.23E+06
20 PRINT A%
```

```
RUN
Overflow In 10
Ready
```

La conversión de valores reales a enteros se lleva a cabo tanto durante la ejecución de sentencias de asignación, como durante la evaluación de funciones y sentencias. En concreto, este tipo de conversión se realiza cuando:

- Se utiliza un número real como subíndice en una matriz.
- Se utilizan números reales como caracteres o coordenadas gráficas.
- Se utiliza un número real como número de fichero.
- Se utiliza un número real como número de tecla de función programable.
- Se utiliza un número real en una sentencia que requiere un número entero como operando.
- Se utiliza un número real para llamar a una función que requiere un número entero como argumento.

- (3) Cuando a una variable numérica de una determinada precisión se le asigna un dato numérico de menor precisión, éste se convierte a la precisión de la variable. No obstante, el dato resultante no se vuelve en realidad más preciso. Por ejemplo, si se asigna el dato B de simple precisión a la variable A# de doble precisión, solamente los primeros 6 dígitos de la variable A# son fiables, ya que el dato original B tenía una precisión de sólo 6 dígitos.

El error relativo en la conversión de precisión simple a doble se puede expresar como sigue:

$$\text{Error relativo} = \text{ABS}((A\# - B)/B) < 5.96\text{E-}08$$

Es decir, el valor absoluto de la diferencia entre los números en simple precisión y doble precisión, dividido por el número de simple precisión es siempre menor que 5.96E-08.

(Ejemplo)

```
10 B=3.02
20 A#=B
30 PRINT B,A#

RUN
   3.02      3.019999980926514
Ready
```

- (4) Cuando se realiza una operación relacional o aritmética con números en niveles de precisión distintos, el operando en el nivel de precisión inferior se convierte al nivel de precisión superior. El resultado de la operación está en el nivel de precisión superior.

(Ejemplo 1)

```
10 A#=7#/6
20 B#=7#/6
30 PRINT A# ,B#
```

RUN

1.1666666666666667

1.1666666666666667

Ready

La división 7#/6 se realiza con doble precisión. Se asigna el resultado a A# como un número de doble precisión.

(Ejemplo 2)

```
10 A=7#/6
20 PRINT A
```

RUN

1.16667

Ready

La división 7#/6 se realiza con doble precisión, pero se redondea el resultado para conseguir un número de simple precisión puesto que A es una variable de simple precisión.

- (5) En las operaciones lógicas, todos los valores de los operandos se convierten en enteros. Los resultados son también enteros. Los enteros que resultan de esta conversión han de estar comprendidos entre -32768 a 32767; de lo contrario, se genera un error de desbordamiento.

(Ejemplo 1)

```
10 A # = 34.56
20 B=NOT A#
30 PRINT B,A #
```

RUN

-36

34.56000137329102

Ready

(Ejemplo 2)

```
10 A# = 34.56D7  
20 B = NOT A #  
30 PRINT B,A #
```

```
RUN  
Overflow In 20  
Ready
```

2.7. Expresiones

Una expresión es una fórmula para una operación. Se compone de una o más constantes, variables y/o funciones que pueden o no estar unidas por operadores. El resultado de la evaluación de una expresión es un número o una cadena. Las expresiones se clasifican como sigue:

- Expresiones aritméticas
- Expresiones de relación
- Expresiones lógicas
- Expresiones con cadenas de caracteres

Las tres primeras se llaman expresiones numéricas, ya que los resultados de su evaluación son numéricos.

2.7.1. Expresiones aritméticas

Una expresión aritmética es cualquier combinación de datos numéricos (variables, constantes, y/o funciones) que están unidos por operadores aritméticos. Los datos numéricos sin operadores aritméticos también son expresiones aritméticas.

Operador	Función	Notación
Aritmético		
+	Suma	$A+B$
-	Resta	$A-B$
*	Multiplicación	$A*B$
/	Division en coma flotante	A/B
^	Exponenciación	A^B
\	División entera. (El cociente es la parte entera del cociente obtenido por la división en coma flotante.)	$A \setminus B$
MOD	El resto que queda después de dividir por un entero.	$A \text{ MOD } B$

En la división “entera” (\) y en la operación de módulo (MOD), los operandos no enteros se redondean antes de la operación.

(Ejemplos)

$$24.35 \setminus 6.87 = 24 \setminus 7 = 3$$

$$10.2 \text{MOD} 4 = 10 \text{MOD} 4 = 2$$

2.7.2. Expresiones relacionales

Una expresión de relación se utiliza para comparar dos valores. Si el resultado es cierto, el valor de la expresión es -1; si es falso, el valor es 0.

Operador de Relación	Función	Notación
=	Igual a	$A = B$
<>, ><	Distinto de	$A < > B$
<	Menor que	$A < B$
>	Mayor que	$A > B$
<=, =<	Menor o igual que	$A <= B$
>=, =>	Mayor o igual que	$A >= B$

(1) Comparación entre números

Una comparación numérica puede utilizarse en una sentencia IF para controlar el flujo del programa.

(Ejemplo 1)

```
10 A=10:B=20
20 IF A>B THEN PRINT "A>B"
    ELSE PRINT "A<B"
```

```
RUN
A<B
Ready
```

(Ejemplo 2)

```
10 A=10<20:B=10>20
20 PRINT A:B
```

```
RUN
-1 0
Ready
```

(2) Comparación de caracteres

En la comparación de cadenas de caracteres, éstos se comparan según sus códigos. (Ver Apéndice B.). Dos cadenas de caracteres son iguales si todos los códigos de los correspondientes caracteres son iguales. Si algún carácter de una cadena es diferente del de su correspondiente en la otra cadena, la cadena que contiene el carácter con código de menor valor se considera menor que la otra. Si durante la comparación una de las cadenas se termina, esta última es menor que la otra. Hay que tener en cuenta que también se comparan los caracteres blancos de las cadenas de caracteres.

(Ejemplos)

"ABC" = "ABC"

"AAB" < "AAC"

"A&" > "A#"

"PR " > "PR"

"cm" > "CM"

"LOAD" < "LOADM"

A\$ < "02:37:15" (si A\$ = "01:25:48")

2.7.3. Expresiones lógicas

Las expresiones lógicas se usan en operaciones lógicas de bit, y para evaluar al mismo tiempo varias expresiones de relación. Una expresión lógica está formada por operadores lógicos y datos numéricos (variables, constantes, y/o funciones). Los datos numéricos de tipo no entero se redondean antes de la operación. Los operadores lógicos son NOT, AND, OR, XOR, IMP y EQV, y cada operación se realiza en unidades de bit. A continuación se muestran los resultados de cada operación lógica de bit.

(1) NOT (negación)

X	NOT X
1	0
0	1

(2) AND(intersección lógica)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

(3) OR (unión lógica)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

(4) XOR (OR exclusivo)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

(5) IMP (implicación)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

(6) EQV (equivalencia)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

(Ejemplos)

(1) NOT (negación)

A% = NOT 1

Ya que $1 = (0000000000000001)_2$, NOT 1 = $(1111111111111110)_2 = -2$

(2) AND (intersección lógica)

B%=2 AND 3

Ya que 2 = (00000000000000010)2

(00000000000000011)2

B% = (00000000000000010)2 = 2

y 3 =

(3) OR (unión lógica)

C%=-1 OR -4

Ya que -1 = (1111111111111111)2

(1111111111111100)2

C% = (1111111111111111)2 = -1

y -4 =

(4) XOR (OR exclusivo)

D%=4 XOR -3

Ya que 4 = (0000000000000100)2

(1111111111111101)2

D% = (1111111111111100)2 = -7

y -3 =

(5) IMP (implicación)

E%=1 IMP 3

Ya que 1 = (0000000000000001)2

(0000000000000011)2

E% = (1111111111111111)2 = -1

y 3 =

(6) EQV (equivalencia)

F%=5 EQV -3

Ya que 5 = (0000000000000101)2

(1111111111111101)2

F% = (0000000000000111)2 = 7

y -3 =

2.7.4. Expresiones cadena

Una expresión con una o más variables cadena y/o constantes cadena, concatenadas con el operador más (+) se llama expresión de cadena. El operador más (+) utilizado en una expresión de cadena enlaza dos cadenas de caracteres; no realiza suma aritmética.

(Ejemplos)

```
10 A$="FILE"  
20 B$="NAME"  
30 PRINT A$ B$ "=";
```

```
RUN  
FILENAME=
```

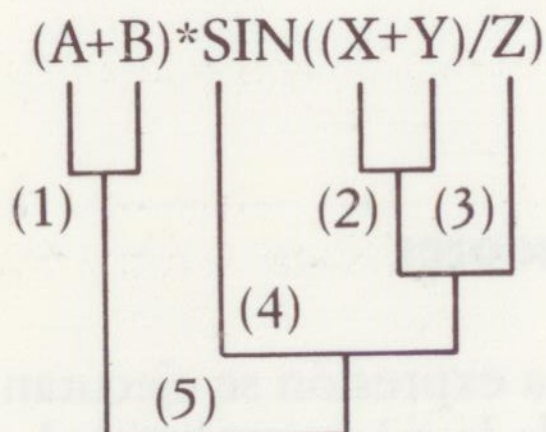
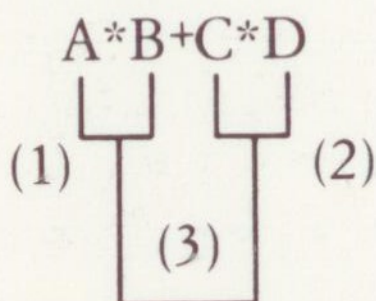
2.7.5. Prioridad de operadores

Las operaciones dentro de una expresión se ejecutan con la siguiente prioridad. Con operaciones de la misma prioridad, el orden es de izquierda a derecha.

Prioridad	Operador
1	^
2	- (negación)
3	*, /
4	\
5	MOD
6	+, -
7	operadores relacionales
8	NOT
9	AND
10	OR
11	XOR
12	EQV
13	IMP

El orden de prioridad de evaluación puede cambiarse utilizando paréntesis. Las expresiones entre paréntesis se evalúan, en primer lugar, comenzando por el juego de paréntesis más interior y continuando hacia el más exterior.

(Ejemplos)



2.8. Descriptor de fichero

En el F-BASIC, todos los dispositivos de entrada/salida (E/S) se manejan utilizando el concepto de fichero. Un fichero es una colección de datos lógicos en un medio de entrada/salida. Existen dos tipos de ficheros en F-BASIC: ficheros de programas y ficheros de datos. Cada fichero se identifica por un descriptor de fichero.

(1) Número de fichero

Las operaciones de entrada/salida se llevan a cabo utilizando los números de fichero. Un número de fichero es un número lógico asignado a un descriptor de fichero, y se define en una sentencia OPEN. El número de fichero debe estar entre 1 y 16, y puede estar representado por una constante, una variable o una expresión.

(2) Formato del descriptor de fichero

El descriptor de fichero consta de las siguientes cadenas:

“[nombre de dispositivo][[(opciones)][nombre de fichero]]”

Algunos de los nombres de fichero y de dispositivo pueden omitirse. El descriptor de fichero ha de especificarse entre comillas (“), tal como se indica en el formato de descripción. Puede especificarse con una variable cadena o una expresión cadena.

(3) Nombres de dispositivos

Se usan los siguientes nombres de dispositivos de entrada/salida en el F-BASIC.

Nombre de Dispositivo de entrada/salida	Nombre de Dispositivo	Entrada	Salida	Comentarios
Teclado	KYBD:	Si	No	Los puertos 1 a 4 están en la unidad de expansión
Pantalla	SCRN:	No	Si	
Impresora	LPT0:	No	Si	
RS-232C:				
Puerto 0	COM0:	Si	Si	
1	COM1:	Si	Si	
2	COM2:	Si	Si	
3	COM3:	Si	Si	
4	COM4:	Si	Si	
Cinta de cassette	CAS0:	Si	Si	Dispositivo por omisión en modo ROM
Disquete flexible 0	0:	Si	Si	Dispositivo por omisión en modo DISCO.
1	1:	Si	Si	
2	2:	Si	Si	
3	3:	Si	Si	

(4) Opción

Las funciones de comunicación del RS-232C pueden establecerse especificando 6 o menos caracteres alfanuméricos entre paréntesis.

(5) Nombre de fichero

El nombre de fichero puede tener una longitud de hasta 8 caracteres, y no debe incluir un signo de dos puntos (:), comillas (") o paréntesis. Si el fichero está en un cassette de burbuja o en disco, su nombre de fichero no puede ser un nombre de fichero nulo. El nombre de dispositivo que precede a un nombre de fichero, identifica la unidad en la que está presente el fichero. Si el nombre del dispositivo se omite, se tomará "CAS0:" en modo ROM y "0:" en modo DISCO.

(6) Ejemplos de descriptores de fichero

"CAS0:FILE1"	Cassette de cinta
"FILE1"	Cassette de cinta, valor por omisión en modo ROM.
"0:FILE2"	Disquete flexible 0
"FILE2"	Disquete flexible 0, valor por omisión en modo DISCO.
"COM0:(S8N2)"	Puerto 0 de la RS-232C

2.9. Inicialización

Al encender el sistema o cuando se pulsa el botón de reset, se inicializan los puertos de comunicación serie RS-232C, las teclas de función programable y el área de cadena, del siguiente modo:

(1) Pantalla

Tamaño de pantalla:	40 (caracteres) x 20 (líneas)
Primera línea:	Línea 0
Número de líneas:	20
Visualización de teclas de función programables:	Ninguna
Color en pantalla:	Blanco

Estos estados se pueden cambiar con las sentencias WIDTH y CONSOLE.

(2) Puertos RS-232C

Reloj:	Reloj lento (divisiones 1/64)
Longitud de palabra:	8 bits
Paridad:	Sin paridad
Bits de parada:	2 bits
Modo:	Comunicación full dúplex
Salto de línea automático:	No activado

Estos estados se pueden cambiar con la sentencia TERM.

(3) Teclas de función programables

Se define inicialmente cada tecla de función programable del siguiente modo:

F-BASIC V1.0 y V2.0 (en modo ROM o modo de DISCO)

PF1	AUTO_
PF2	LIST C _R
PF3	RUN C _R
PF4	GO_TO_
PF5	CONT C _R
PF6	?DATE\$,TIME\$ C _R
PF7	LIST"LPT0: C _R
PF8	KEY_
PF9	LOAD_
PF10	HARDC C _R

El símbolo "_" indica un carácter en blanco. El símbolo C_R indica un retorno de carro.

F-BASIC V3.0

El estado inicial depende del modo empleado.

Tecla de función programable	Modo ROM	Modo DISCO
PF1	AUTO_	AUTO_
PF2	LIST C _R	LIST C _R
PF3	RUN C _R	RUN C _R
PF4	CONT C _R	CONT C _R
PF5	LLIST C _R	LLIST C _R
PF6	LOAD C _R	LOAD“
PF7	SAVE“	SAVE“
PF8	?DATE\$,TIME\$ C _R	FILES
PF9	SCREEN7,7 C _R	SCREEN7,7 C _R
PF10	HARDC C _R	HARDC C _R

El símbolo _ indica un carácter en blanco. El símbolo C_R indica un retorno de carro.

(4) Area de cadenas de caracteres

Este área se usa para almacenar las expresiones cadena. Su tamaño es inicialmente de 300 bytes, pero puede cambiarse mediante el comando CLEAR.

2.10. Gestión y estructura de ficheros

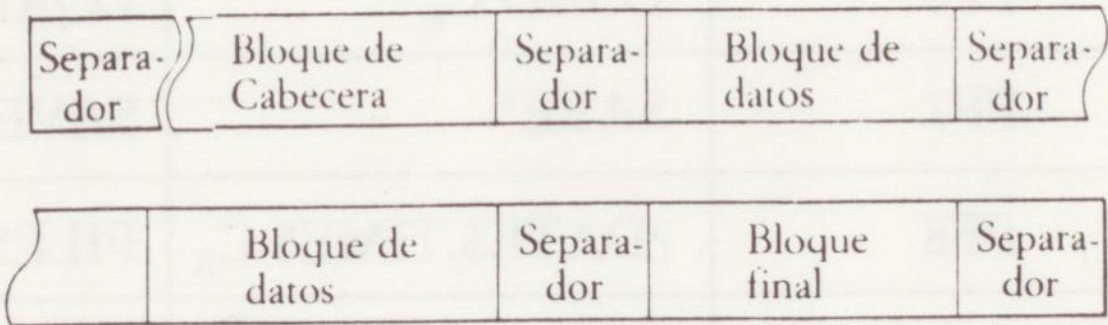
El F-BASIC maneja entradas y salidas de datos utilizando ficheros. Un fichero es un conjunto significativo de datos en un medio de entrada/salida. Un fichero consta de los datos que se sacan desde que se abre el fichero bajo un nombre definido por el usuario, hasta que se cierra dicho fichero.

Esta sección describe la estructura y manejo de grabadores de datos, cassettes de burbuja, minidisquetes flexibles y disquetes flexibles tipo estándar.

2.10.1. Grabador de datos

El F-BASIC graba datos en un fichero de grabador de datos, con el siguiente formato:

Formato de fichero



Formato de bloque

	01	3C	Tipo de bloque	Longitud del bloque	Datos	Suma de prueba	
--	----	----	-------------------	------------------------	-------	-------------------	--

Tipo de bloque

00 bloque de cabecera

01 bloque de datos

FF bloque final

Longitud del bloque

Indica la longitud de los datos en bytes (00 a FF).

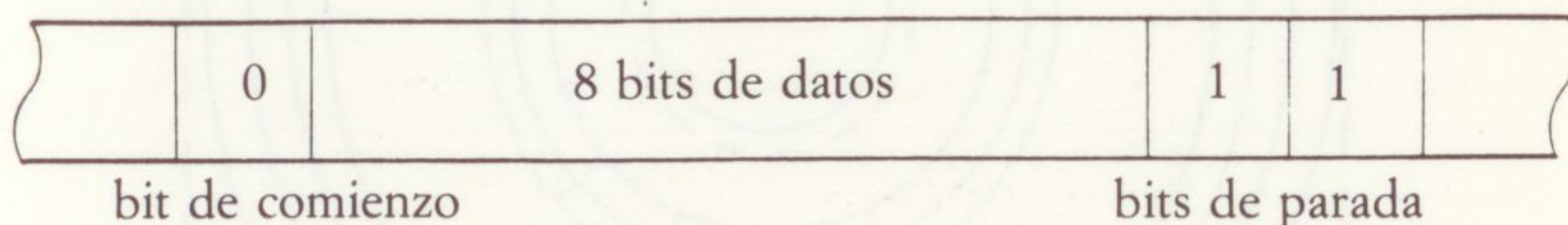
Bloque de datos

Contiene de 0 a 255 bytes de datos. El bloque de cabecera contiene nombre de fichero, organización de fichero y atributo (ASCII/binario). El bloque final no contiene ningún dato.

Suma de Prueba

Proporciona una forma de verificar el tipo de bloque, la longitud del bloque y los datos.

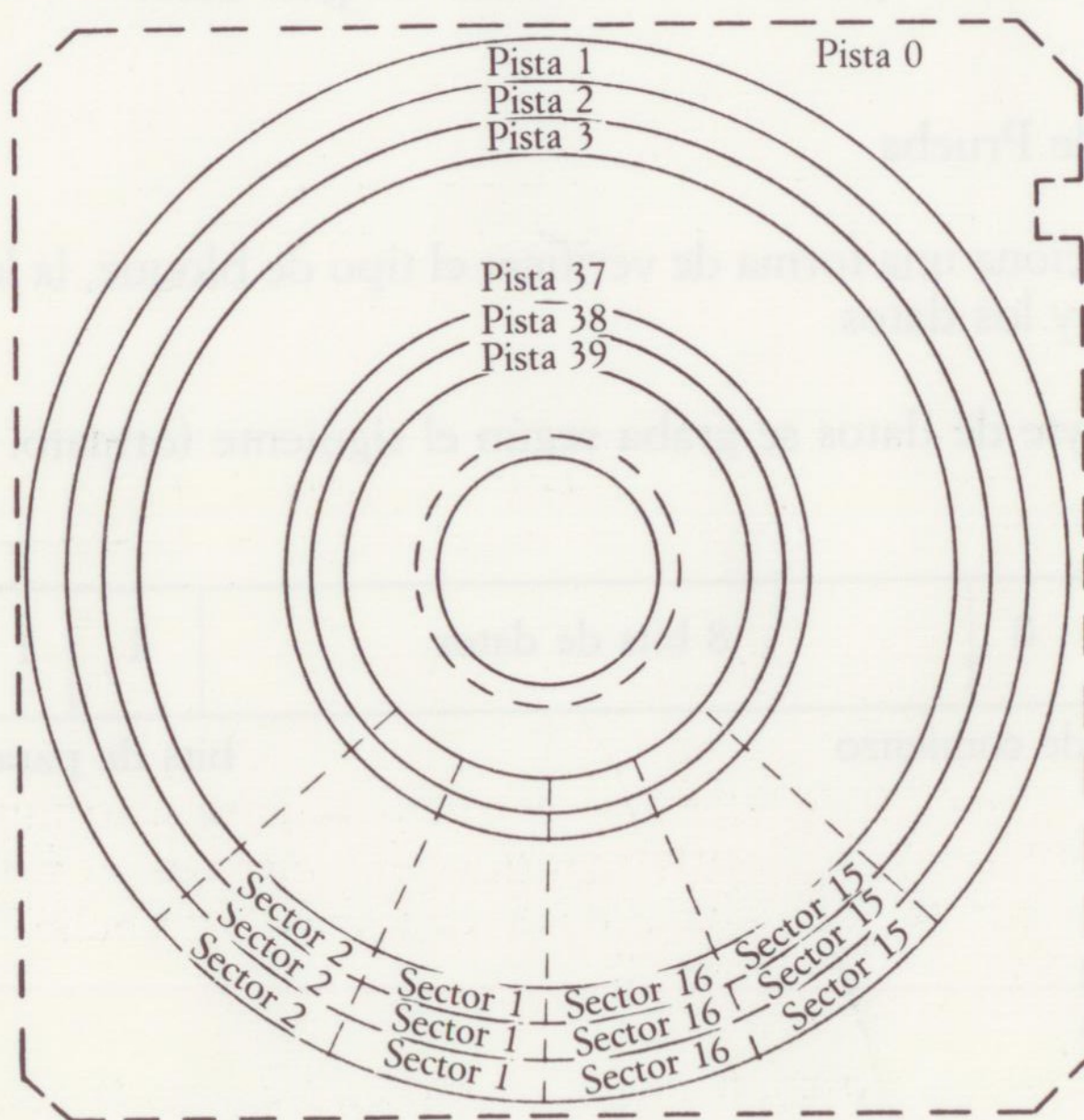
Cada byte de datos se graba según el siguiente formato.



2.10.2. Disquete flexible

(Estructura)

El minidisquete flexible que se puede utilizar en el F-BASIC es un disco de 5,25 pulgadas, de doble cara y doble densidad. Tiene la siguiente estructura:



Una de las caras del disco se llama cara 0, y la otra cara 1. Las principales especificaciones del disco se detallan a continuación.

Capacidad de datos/disco	320K bytes
Número de cilindros/disco	40
Número de pistas/disco	80 (40 x 2)
Número de sectores/pista	16
Longitud de sector en bytes	256

(Dirección de sector)

Los números de pista y de sector se utilizan para identificar los sectores en un disco. Puede que haga falta también un número de cara para los discos de doble cara. No obstante, el F-BASIC asigna números de serie a todos los sectores en ambas caras, y los identifica por número de serie y número de pista. La relación entre el número de serie y el número de sector físico se muestra a continuación.

Cara 0			Cara 1			
Sector 1	16	1	16			
Pista 0	Sector 1	-----	Sector 16	Sector 17	----	Sector 32
Pista 39	Sector 1	-----	Sector 16	Sector 17	----	Sector 32

(Grupo)

En los minidisquetes flexibles, un sector es la unidad mínima de datos para acceso a lectura/escritura. Sin embargo, el F-BASIC usa ocho sectores como unidad mínima de trabajo. Los ocho sectores se llaman un grupo (CLUSTER).

El Grupo 0 está formado por los sectores 1 a 8 en la pista 2, y el grupo 151 por los sectores 25 a 32 en la pista 39.

(Asignación de Pistas)

El F-BASIC usa la siguiente asignación de pistas para el minidisquete flexible.

Número de pista	Disco del sistema	Disco del usuario
0	IPL, ID	ID
1	FAT, directorio	FAT, directorio
2 a 6	Programa del sistema	AREA DE USUARIO
7 a 39	Area del usuario	

IPL (Initial Program Loader o cargador inicial de programa) es un programa que carga los programas del disco en la memoria. ID (identificación) es un campo que indica si el disco puede ser usado o no por el F-BASIC. El código DISK contiene instrucciones relacionadas con el disco. Cuando se activa el F-BASIC, los códigos de disco se cargan en memoria, comenzando por la dirección &H7000. La tabla de asignación de los ficheros (FAT) indica como se usan los grupos. El directorio es una lista de los nombres, tipos y posiciones de ficheros. El área de usuario está a disposición del mismo para el almacenamiento de ficheros.

(Estructura de las pistas 0 y 1)

Pista 0

Sector	1	2	3	4	5	6	7	8	12	13	14	15	16
Cara 0	I P L	ID	Reservados						Reservados				
Sector	17	18	19	20	21	22	23	24	28	29	30	31	32
Cara 1	Reservados								Reservados				

Pista 1

Sector	1	2	3	4	5	6	7	8	12	13	14	15	16
Cara 0	FAT	Reservados		Directorio					Directorio				
Sector	17	18	19	20	21	22	23	24	28	29	30	31	32
Cara 1	Directorio								Directorio				

(FAT)

La tabla FAT indica como se usan los grupos (CLUSTERS). La utiliza el F-BASIC para gestionar el espacio del disco. Esta tabla, está grabada en el primer sector de la pista 1. Los bytes 6 a 157 de ese primer sector corresponden a los grupos 0 a 151. El significado de los datos contenidos en cada byte es el siguiente.

Datos (en hexadecimal)	Significado
0 a 97	Se está utilizando el grupo, y le sigue otro grupo que se identifica por el número indicado.
C0 a C7	Se está usando el grupo. Es el último del fichero. El resultado de restar BF del valor indicado representa el número de sectores realmente usados en el grupo.
FD	No se está usando ningún sector del grupo.
FE	El grupo es un área del sistema.
FF	El grupo no se ha usado.

(Directorio)

El directorio está en los sectores 4 al 32 de la pista 1. Cada sector del directorio se divide en 8 bloques de 32 bytes; a cada bloque se le denomina una franja de directorio, y está formada como muestra la siguiente tabla.

Posición del byte en cada franja de directorio	Contenidos
0 a 7	Nombre del fichero
8 a 10	Reservados
11	Tipo de fichero 00: programa fuente de BASIC 01: datos en BASIC 02: fichero de lenguaje máquina
12	Indicador ASCII/Binario FF: ASCII 00: Binario
13	Indicador de acceso aleatorio
14	Número de grupo de comienzo del fichero
15 a 31	Reservados

Existen 232 franjas de directorio, pero el número real de ficheros que pueden catalogarse se limita a 152, puesto que el número de grupos disponibles es 152.

3. INSTRUCCIONES EN F-BASIC

Cómo leer este capítulo

Este capítulo describe las instrucciones en F-BASIC de las tres versiones. Cada instrucción se describe bajo los apartados: (Función), (Formato), (Versiones), (Descripción) y (Ejemplo(s)).

La instrucción va seguida de la traducción al español entre paréntesis.

Ejemplo: REM (remark = comentario)

(Función)

Se da una breve descripción de la instrucción.

(Formato)

Se da el formato de codificación de la instrucción. Se usa la siguiente notación:

- (1) Los caracteres alfabéticos en la instrucción pueden introducirse en mayúsculas o minúsculas. No obstante, una cadena de caracteres que aparece encerrada entre comillas (") debe ser introducida exactamente como se indica.
- (2) Las partes entre corchetes [] son optativas. Si se omiten, se usa el valor previo o el valor por omisión.
- (3) Cuando se encierra un grupo de elementos entre llaves debe elegirse uno de ellos.

- (4) Tres puntos suspensivos significan que el elemento precedente puede repetirse dentro de la línea.

Ejemplo: READ nombre de variable [,nombre de variable]...
significa que se puede especificar de la forma siguiente:

READ A, B, C, D

- (5) Códigos especiales que no sean corchetes, tales como paréntesis, coma (,), punto y coma (;), guión (-), signo igual (=), y comillas (") se deben codificar en la posición especificada.
- (6) La marca "-" indica un espacio.
- (7) El código de la paleta de colores en la Sección 3.4 es el código de color en las versiones V1.0 y V2.0.

(Versiones)

La sentencia puede utilizarse en las versiones que aparezcan recuadradas.

V1.0

Puede usarse en F-BASIC V1.0.

V2.0

Puede usarse en F-BASIC V2.0.

V3.0

Puede usarse en F-BASIC V3.0.

Las sentencias marcadas (Disk) sólo operan en el modo DISCO en

V1.0 y **V3.0**

(Descripción)

Se da una explicación detallada, con notas, de la sentencia. El código de la paleta de colores en la Sección 3.4 es el código de color en la V1.0 y V2.0.

(Ejemplo(s))

Se detallan uno o más programas como ejemplo.

Modo ROM y modo DISCO

Las sentencias de que dispone el F-BASIC V1.0 y V3.0 dependen de si el sistema opera en el modo ROM o en el modo DISCO. Para la versión V2.0, sólo se puede utilizar el modo DISCO.

En el modo ROM, solamente pueden usarse aquellas sentencias que se hayan almacenado en ROM como "firmware". En el modo DISCO, se pueden usar sentencias adicionales introducidas en la RAM desde un disco.

Modo de Operación

El intérprete BASIC se activa en cuanto se enciende el sistema principal. Empieza por visualizar un título y la palabra "Ready" (preparado). "Ready" indica que el BASIC está en el nivel de comandos, preparado para aceptar comandos. El usuario puede acceder al BASIC en el modo directo, indirecto o terminal.

En el modo directo, los comandos o sentencias en BASIC se introducen sin números de línea y se ejecutan inmediatamente. Cuando finaliza la ejecución, el BASIC vuelve al nivel de comandos y visualiza de nuevo la palabra "Ready". Los comandos o sentencias no se guardan en memoria.

En el modo indirecto, se introducen programas. La ejecución no se realiza inmediatamente, sino que el programa se almacena en el área de programas BASIC con las líneas en orden numérico. El programa almacenado puede ejecutarse por un comando RUN o mediante las sentencias GOTO o GOSUB. Para introducir una línea de programa, primero hay que especificar un número de línea, y después dar uno o más comandos o sentencias. Después de introducir la línea, el BASIC sigue en el nivel de comandos, aunque no visualice la palabra "Ready".

En el modo terminal, el FM-7 puede usarse como un terminal con otro ordenador. Si se introduce el comando TERM, el BASIC se adapta al modo terminal.

Modo de Operación

3.1. Comandos

3.1.1. AUTO (auto)

(Función)

El comando AUTO genera automáticamente números de línea para cada línea.

(Formato)

AUTO[número de línea][[,incremento]]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Los números de línea se asignan en orden ascendente, empezando por el número de línea que se especifica y aumentando para cada nueva línea según el valor indicado en incremento.
- El número de línea y el incremento deben ser enteros de cinco dígitos o menos. El valor por defecto tanto para el número de línea como para el incremento es 10.
- En un programa no se puede especificar un número de línea mayor de 63999 ni un número de línea que ya ha sido generado.
- Se puede anular la función AUTO y restaurar el BASIC al nivel de comandos al pulsar dos veces la tecla RETURN. También se puede pulsar **[control] [C]** , **[control] [X]** , o la tecla BREAK (tecla STOP en la versión **V1.0** y **V2.0**)

- Mientras se escribe un programa usando la función AUTO, sólo se puede corregir la línea en curso; no se pueden corregir las líneas ya introducidas. Tampoco se puede editar el programa directamente en pantalla mientras el comando AUTO sea válido. Para hacer una corrección de este tipo, primero hay que cancelar el AUTO, y después introducir un comando LIST para visualizar la línea o líneas que se vayan a corregir en pantalla.

(Ejemplos)

AUTO 10

Los números de línea se asignan empezando por 10 y aumentando en 10 por cada línea: 10, 20, 30, ...

AUTO 200,45

Los números de línea se asignan empezando por 200 y aumentando en 45 por cada línea: 200, 245, 290, ...

AUTO ,30

Los números de línea se asignan empezando por 10 y aumentando en 30 por cada línea: 10, 40, 70,...

AUTO

Los números de línea se asignan empezando por 10 y aumentando en 10 por cada línea: 10, 20, 30,...

AUTO 300,

Los números de línea se asignan empezando por 300 y aumentando en el número especificado en la última sentencia AUTO utilizada: Por ejemplo, si la sentencia especifica un incremento de 20, los números de línea son 300, 320, 340,... Al usar el valor por defecto, los números de línea son 300, 310, 320,...

3.1.2. DELETE (suprimir)

(Función)

El comando DELETE suprime líneas del programa.

(Formato)

DELETE[número de línea 1][{,-}[número de línea 2]]

(Versiones)

V1.0 V2.0 V3.0

(Descripción)

- Si se especifican el número de línea 1 y número de línea 2, dichas líneas, junto con todas las que hay entre ellas, se suprimen.
- Si se especifica solamente el número de línea 1, sólo se suprime dicha línea.
- Si se especifica una coma (,) o un guión (-) y la línea número 2, se suprimen todas las líneas desde el principio hasta la línea 2.
- Si se especifica el número de línea 1 y la coma (,) o el guión (-), se suprimen todas las líneas desde la línea número 1 hasta el final.
- Si sólo se especifica la coma (,) o el guión (-), se suprimen todas las líneas.
- Si aparece una sentencia DELETE en un programa, el BASIC la ejecuta y vuelve al nivel de comandos.

(Ejemplos)

DELETE 40,500 o DELETE 40-500

Se suprimen las líneas 40 a 500.

DELETE ,200 o DELETE -200

Se suprimen las líneas desde la primera hasta la 200.

DELETE 120, o DELETE 120-

Se suprimen las líneas desde la 120 hasta el final.

DELETE 95

Se suprime la línea 95.

DELETE,

Se suprimen todas las líneas.

3.1.3. LIST (listar)

(Formato 1)

LIST [número de línea 1] [{,- [número de línea 2]]

(Abreviado L.)

(Función)

El comando LIST con el formato 1 lista todo o parte del programa en memoria.

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Si no se especifica número de línea, se lista el programa completo.
- Si se especifica solamente el número de línea 1, solo se lista dicha línea.

- Si se especifica el número de línea 1 y la coma (,) o el guión (-), se listan todas las líneas desde la línea número 1 hasta el final.
- Si se especifica una coma (,) o un guión (-) y la línea número 2, se listan todas las líneas desde el principio hasta la línea número 2.
- Si se especifican el número de línea 1 y número de línea 2, dichas líneas, junto con todas las que hay entre ellas, se listan.
- Cuando se produce un error, se puede usar "LIST." para ver la línea que contiene el error.
- Si aparece una sentencia LIST en un programa, el BASIC la ejecuta y vuelve al nivel de comandos, sin procesar el resto del programa.

(Ejemplos)

LIST

Se listan todas las líneas.

LIST 50

Se lista la línea 50.

LIST 10-300 ó LIST 10,300

Se listan las líneas desde la 10 hasta 300.

LIST -125 ó LIST ,125

Se listan las líneas desde la primera hasta la 125.

LIST 250- ó LIST 250,

Se listan las líneas desde la 250 hasta el final.

(Formato 2)

LIST "Descriptor de fichero" [, [número de línea 1] [{ , } [número de línea 2]]]]

(Función)

El comando LIST en formato 2 copia todo o parte de un programa desde la memoria a un fichero especificado.

El significado de los operandos es el mismo que en el formato 1. Especificando cassette, disco flexible o cassette de burbuja como dispositivo de entrada/salida, produce el mismo efecto que SAVE en formato ASCII.

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

La expresión cadena o la variable cadena que se use para especificar el descriptor del fichero debe estar precedida por comillas ("); por lo tanto, conectar una cadena vacía al principio de la expresión cadena que empiece con, o consiste en, una variable cadena.

(Ejemplos)

LIST "LPT0:" 300,1250

Las líneas 300 a 1250 se sacan a la impresora.

LIST "CAS0:"

Todas las líneas se sacan al cassette. Las especificaciones de los números de línea son iguales que en el formato 1.

3.1.4. UNLIST (no listar)

(Función)

El comando UNLIST suprime el listado de las líneas especificadas.

(Formato)

UNLIST número de línea

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Después de la ejecución de este comando, se suprime la visualización de cualquier línea siguiente de número más alto. Una vez ejecutado el comando UNLIST, no se pueden listar las líneas que siguen a la línea especificada. Al almacenar un programa después de ejecutar este comando, debe especificarse SAVE en forma binaria, y no en la forma ASCII. De otro modo, no pueden guardarse las líneas afectadas por este comando.

(Ejemplo)

```
10 UNLIST 20
20 A=10
30 B=20
40 C=A+B
50 PRINT C
60 END
```

```
Ready
RUN
30
```

```
Ready
LIST
```

```
10 UNLIST 20
Ready
```

Después de ejecutar el comando UNLIST en la línea 1, no se visualiza la línea 20 ni las siguientes.

3.1.5. LLIST (listar por la impresora)

(Función)

El comando LLIST saca por impresora todo o parte de un programa.

(Formato)

LLIST [número de línea 1][{,}[número de línea 2]]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Los números de línea se especifican de la misma manera que para LIST. LLIST y LIST "LPT0:" son comandos equivalentes.
- Cuando una sentencia LLIST aparece en un programa, el BASIC la ejecuta y vuelve al nivel de comandos.

(Ejemplos)

LLIST

Todas las líneas se sacan por la impresora.

LLIST 200

Sólo la línea 200 se saca por la impresora.

LLIST 30-200 o LLIST 30,200

Las líneas 30 a 200 se sacan por la impresora.

LLIST -55 o LLIST ,55

Las líneas desde la primera hasta la 55 se sacan por la impresora.

LIST 360- o LLIST 360,

Las líneas desde 360 hasta el final se sacan por la impresora.

3.1.6. RENUM (renumerar)

(Función)

El comando RENUM renumera las líneas de programa.

(Formato)

RENUM[número de línea nuevo][,[número de línea viejo]
[,incremento]

(Versiones)



(Descripción)

- El número de línea nuevo indica el número deseado para la línea inicial. El valor por omisión es 10.
- El número de línea viejo indica el número de línea del programa en el cual se comienza la renumeración. El valor por defecto es el primer número de línea del programa.
- El incremento indica el intervalo entre números de línea nuevos. El valor por omisión es 10.
- El comando RENUM corrige toda referencia de número de línea (en GOTO, GOSUB, THEN, ON GOTO, ON GOSUB, ERL y otras sentencias parecidas) a los nuevos números de línea correspondientes. Si falta en el programa un número de línea viejo a que se hace referencia en alguna parte, entonces se visualiza el mensaje de error "Undefined Line Number XXXXX en YYYYYY" = número de línea sin definir XXXXX en YYYYYY (donde XXXXX representa el número de línea viejo que falta e YYYYYY representa el nuevo número de línea correspondiente), y no se cambia la referencia al número de línea no existente.
- Cuando una sentencia RENUM aparece en un programa, el BASIC la ejecuta y vuelve al nivel de comandos.

· RENUM no produce números de línea mayores de 63999 ni cambia el orden de las sentencias de programa. Por ejemplo, si los números de línea 10, 20, y 30 aparecen en un programa, RENUM 15, 30 produciría un error de "Illegal Function Call" (llamada ilegal a la función).

(Ejemplos)

RENUM

Este comando renumera todas las líneas de programa, asignando 10 al primer número y aumentando los sucesivos números de línea en 10.

RENUM 30,50,20

NOTA: No se puede renumerar una línea con un número menor de línea que los números ya empleados.

La línea 50 se renumera como 30, y las líneas siguientes se enumeran con incrementos de 20 de modo que siguen 50, 70, etc.

RENUM ,20, 10

La línea 20 se renumera con 10, y las líneas siguientes se renumeran en incrementos de 10, de modo que continúan 20, 30, 40, etc.

RENUM 20,,20

Este comando renumera todas las líneas del programa, asignando el número 20 a la primera línea y aumentando los siguientes números en 20, de modo que continúan 20, 40, 60, etc.

RENUM 30,15

La línea 15 se renumera como 30, y las líneas siguientes se renumeran en incrementos de 10 de forma que continúan 30, 40, 50, etc.

RENUM ,30

La primera línea del programa se renumera como 10, y las líneas siguientes se renumeran en incrementos de 30, de forma que continúan 10, 40, 70, etc.

RENUM, 50

La línea 50 se renumera como 10. Las líneas siguientes se renumeran en incrementos de 10, de forma que continúan 20, 30, 40, etc.

3.1.7. NEW (nuevo)

(Función)

El comando NEW elimina de la memoria todas las sentencias de programa existentes e inicializa todas las variables.

(Formato)

NEW

(Versiones)

V1.0 V2.0 V3.0

(Descripción)

El comando NEW se usa para eliminar de la memoria todas las sentencias de programa existentes antes de introducir un programa nuevo. El BASIC siempre vuelve al nivel de comandos después de la ejecución del comando NEW. Este comando cierra todos los ficheros que puedan estar abiertos.

(Ejemplos)

```
10 FOR I=1 TO 9
20 FOR J=1 TO 9
30 PRINT I;"*";J;"=";J*I
40 NEXT J
50 PRINT
70 NEXT I
80 END
```

Ready
NEW

Ready
LIST

Ready

3.1.8. CLEAR (borrar)

(Función)

El comando CLEAR inicializa todas las variables numéricas a 0 y todas las variables cadena a la cadena vacía, y luego establece la capacidad del área de cadenas de caracteres y/o el límite superior de la memoria usada por el BASIC según especifica el operando (s).

(Formato)

CLEAR[tamaño de área de cadena][,dirección del límite superior de memoria para el BASIC]

(Versiones)

V1.0 **V2.0** **V3.0**

(Descripción)

- Este comando retiene en memoria el programa en curso pero libera todas las áreas de memoria que almacenan datos. Cuando se ejecuta CLEAR, se inicializan todas las variables numéricas a cero, todas las variables cadena a la cadena nula, y se eliminan todas las variables tipo matriz. Se invalida toda información definida por las sentencias DEF (DEFFN, DEFUSR, DEFINT, DEFDBL, DEFSNG, DEFSTR).
- Cuando el BASIC es puesto en marcha se asigna un área de cadenas de 300 bytes como mínimo. Para aumentar este área, a fin de evitar que se produzca un error "Out of String Space" (Espacio de cadenas agotado), puede utilizarse el comando CLEAR.
- El segundo operando fija un límite superior en el área de memoria usada por el BASIC. Cuando se llama a una subrutina de lenguaje máquina desde un programa de BASIC, hay que reservar un área en memoria para ello. Esto se hace especificando una dirección del límite superior de memoria con el segundo operando de una sentencia CLEAR.

(Ejemplos)

`CLEAR 300,&H54FF`

El área de cadenas se fija en 300 bytes, y la dirección de memoria más alta que el BASIC puede usar es la &H54FF.

`CLEAR,&H3000`

El tamaño del área de cadenas permanece inalterado, y la dirección de memoria más alta que el BASIC puede usar fija en &H3000.

`CLEAR 1000`

El área de cadena se fija en 1000 bytes, y la dirección del límite superior de memoria para el BASIC no cambia.

`CLEAR`

El área de cadena y la dirección del límite superior de memoria para el BASIC se inicializan a los valores por defecto.

3.1.9. CONT (continuar)

(Función)

El comando CONT reanuda la ejecución de un programa que se había suspendido después de pulsar la tecla BREAK (tecla STOP para V1.0 y V2.0) o después de ejecutarse una sentencia STOP o END.

(Formato)

CONT

(Versiones)

V1.0 V2.0 V3.0

(Descripción)

- Cuando se introduce un comando CONT en el modo directo después de suspenderse un programa, su ejecución se reanuda a partir de la sentencia siguiente a la que se encontraba cuando fue suspendido. No obstante, si el programa se detiene en una sentencia INPUT o GCURSOR, entonces vuelve a empezar a partir del principio de esa sentencia.
- Después de interrumpir la ejecución de un programa, se pueden comprobar los resultados intermedios y cambiar los valores de las variables introduciendo sentencias en el modo directo. Después, se reanuda la ejecución mediante CONT o GOTO. Sin embargo, no se puede utilizar CONT si se ha modificado el programa durante la interrupción.
- CONT no puede reanudar la ejecución si se pulsa la tecla BREAK (la tecla STOP para V1.0 y V2.0) durante una operación de salida de datos a la impresora, o una operación de entrada/salida al cassette.

(Ejemplo)

```
10 A=123
20 B=456
30 C=A+ B*10
40 STOP
50 PRINT C
60 END
```

```
Ready
RUN
Break in 40
Ready
CONT
4683
```

Ready

3.1.10. RUN (procesar)

(Formato 1)

RUN[número de línea]

(Abreviado R.)

(Función)

El comando RUN en el formato 1 inicia la ejecución de un programa en memoria.

(Versiones)

V1.0 V2.0 V3.0

(Descripción)

- Si se especifica un número de línea, la ejecución del programa empieza a partir de la línea indicada. Si se omite el número de línea, el programa empieza desde la primera línea. Antes de la ejecución, el comando RUN inicializa todas las variables numéricas a 0, y todas las variables cadena a la cadena vacía, y luego cierra todos los ficheros abiertos.
- La ejecución del programa termina cuando se pulsa la tecla BREAK (la tecla STOP en V1.0 y V2.0), o cuando se ejecuta una sentencia END o STOP. Después el BASIC vuelve al nivel de comandos.

(Ejemplos)

RUN

La ejecución del programa empieza a partir de la primera línea.

RUN50

La ejecución del programa empieza a partir de la línea 50.

(Formato 2)

RUN "descriptor de fichero"[R]

(Función)

- El comando RUN en el formato 2 carga un programa en memoria desde un fichero contenido en el disco o cassette especificado, y luego lo ejecuta.

(Versiones)

V1.0/ V2.0/ V3.0/

(Descripción)

- El comando RUN carga un programa en la memoria del ordenador y lo ejecuta después de cerrar todos los ficheros abiertos e inicializar el contenido de la memoria. Especificando R, se puede evitar el cierre de ficheros.
- La expresión cadena o variable cadena que se especifique para el descriptor de fichero, ha de estar delimitada entre comillas (").

(Ejemplos)

RUN "0:PROG"

Un programa con nombre de fichero PROG se carga en memoria desde la unidad de disco 0, y se ejecuta.

RUN "CAS0: PROG1"

Un programa con nombre de fichero PROG1 se carga en memoria desde un cassette, y se ejecuta.

3.1.11. LOAD (cargar)

(Función)

El comando LOAD carga un programa en memoria.

(Formato)

LOAD "descriptor de ficheros"[,R]

(Abreviado LO.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El comando LOAD carga un programa en memoria desde un fichero especificado por un descriptor de fichero. Antes de cargar el programa, cierra todos los ficheros abiertos e inicializa el contenido de la memoria. Si se especifica R, LOAD ejecuta el programa automáticamente sin cerrar los ficheros.

(Ejemplos)

LOAD "CAS0: EX1"

Un programa con nombre de fichero EX1 se carga en memoria desde un cassette.

LOAD "0:EX2"

Un programa con nombre de fichero EX2 se lee desde la unidad de disco "0" y se carga en memoria.

LOAD "1:EX3",R

Un programa con nombre de fichero EX3 se lee desde la unidad de discos "1" se carga en memoria y se ejecuta.

3.1.12. LOAD? (cargar?)

(Función)

El comando LOAD? verifica la suma de prueba en la cinta de cassette.

(Formato)

LOAD?["[CAS0:]nombre de fichero"]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El comando LOAD? hace una comprobación de la igualdad entre el programa de la cinta cassette y el que se ha grabado con SAVE. Si no concuerdan, se visualiza el mensaje de error "Device I/O Error" (Error de dispositivo de entrada/salida).
- Si se omite el descriptor de fichero, se asume el primer fichero que se encuentre en la cinta de cassette.
- CAS0: puede omitirse en el modo ROM.

(Ejemplo)

LOAD? "EX1"

Este comando verifica la suma de prueba con un nombre de fichero EX1 en la cinta de cassette.

3.1.13. SAVE (guardar)

(Función)

El comando SAVE almacena el programa de memoria en un fichero.

(Formato)

SAVE "descriptor de ficheros"[,{P/A}]

(Abreviado SA.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

El programa en memoria se guarda en un fichero especificado por el descriptor de ficheros. Si el descriptor de ficheros especifica un disco, el nombre del fichero debe ser uno nuevo. Si se especifica el nombre de un fichero en el disco, entonces el BASIC reacciona del siguiente modo. En el modo directo, se visualiza el mensaje "Are you sure (Y or N)?" (= Está Vd. seguro? (Sí o No)). Si se contesta con la Y, se elimina el fichero existente, y el programa se guarda en un fichero nuevo. Si se especifica la N, entonces el fichero existente se mantiene en el disco, y el programa en memoria no se salva. En el modo indirecto, el BASIC visualiza el mensaje "File Already Exists In nnnn" (= fichero ya existe en el nnnn) y vuelve al nivel de comandos. nnnn es el número de línea.

Al especificar la opción A, se guarda el programa en forma ASCII (forma de cadena). Sin especificar la A, el programa se guarda en forma binaria (es decir, el programa se codifica internamente). El guardar un programa en forma binaria es más rápido y ahorra espacio de fichero. Sin embargo, el fichero debe ser guardado en formato ASCII si se prevé que el programa va a ser procesado por el comando MERGE en un futuro. Los ficheros guardados en la forma ASCII pueden ser leídos como ficheros de datos sin cambio alguno.

Al especificar P, se protege el contenido del programa. Los ficheros guardados con la opción P producen un error de "Protected Program" (= programa protegido) cuando se cargan y se intenta acceder a líneas de ellos. Dado que la opción P no puede ser borrada de los ficheros, antes de especificarla debe de asegurarse de si es o no necesaria.

(Ejemplos)

SAVE "CAS0: PROG1"

El programa en memoria se guarda en una cinta cassette, en forma binaria, con el nombre de fichero PROG1.

SAVE "0:PROG2",A

El programa en memoria se guarda en la unidad de discos 0, en la forma ASCII, con nombre de fichero PROG2.

SAVE "1:PROG2",P

El programa en memoria se guarda en la unidad de discos 1, en forma binaria, con nombre de fichero PROG2, y está protegido.

3.1.14. FILES (ficheros)

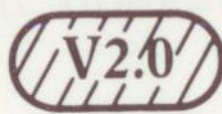
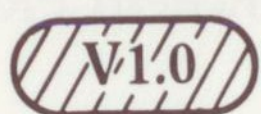
(Función)

El comando FILES visualiza el listado de directorio de un dispositivo especificado.

(Formato)

FILES ["nombre de dispositivo"][,L]

(Versiones)



(Descripción)

- Cada entrada en el directorio contiene la siguiente información:

nombre de fichero	tipo	forma	organización	tamaño
-------------------	------	-------	--------------	--------

Existen tres tipos:

0: fichero de programas en BASIC

1: fichero de datos

2: fichero de programas en lenguaje máquina

Existen dos formas:

A: forma ASCII

B: forma binaria

Existen dos tipos de organización:

S: fichero secuencial

R: fichero de acceso directo

Tamaño indica el número de páginas del fichero, si está contenido en cassette de burbuja, o el número de grupos si está contenido en disco flexible. En cinta de cassette sólo se visualiza el nombre, el archivo y su tipo.

- Al especificar L, se imprime también un listado por la impresora.
- Para terminar una sentencia FILES para un cassette de cinta, pulsar la tecla BREAK (la tecla STOP para V1.0 y V2.0).

(Ejemplos)

FILES "CAS0:"

Los nombres de los ficheros se visualizan en este tipo de formato cuando están contenidos en cassettes.

```
PROG1  0A
PROG2  0B
```

FILES "1:"

Los nombres de los ficheros se visualizan en este tipo de formato cuando están contenidos en el disco flexible 1.

```
EX1      0 A S 1
EX2      0 B S 2
EX3      2 B S 1
EX4      1 A R 3
```

145 Clusters Free

Es decir 145 grupos libres.

3.1.15. NAME (nombrar)

(Función)

El comando NAME cambia el nombre de un fichero en un disquete flexible.

(Formato)

NAME "descriptor de fichero antiguo" AS
"descriptor de fichero nuevo"

(Versiones)



(Descripción)

- El nombre del dispositivo debe ser el mismo para el descriptor del fichero antiguo y del nuevo.

(Ejemplo)

NAME "1:PROG1" AS "1:PROG2"

Este comando cambia el nombre del fichero PROG1 contenido en el disco flexible 1 a PROG2.

3.1.16. KILL (eliminar)

(Función)

- El comando KILL elimina un fichero de un disco flexible.

(Formato)

KILL "descriptor de ficheros"

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Cuando se introduce este comando en el modo directo, se visualiza la pregunta "Are you sure (Y or N)? (Está Vd. seguro? (Sí o No)). Para contestar, se introduce Y para eliminar el fichero, o N para anular el comando. Cuando se usa este comando en el modo indirecto, no se visualiza el mensaje de advertencia, y se elimina inmediatamente el fichero especificado.
- El fichero que se va a eliminar debe estar cerrado. Si se ejecuta un comando KILL con un fichero abierto, se produce un error de "File Already Open" ("Fichero ya abierto").
- Con la versión V3.0, este comando es válido en el modo de disco.

(Ejemplo)

KILL "0:EX2"

Se elimina un fichero llamado EX2 del disquete flexible 0.

3.1.17. MERGE (mezclar)

(Función)

El comando MERGE junta un programa en memoria con otro programa en el fichero especificado.

(Formato)

MERGE "descriptor de ficheros" [,R]

(Abreviado ME.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Si un número de línea se duplica en los dos programas que se van a juntar, la línea del fichero tiene preferencia sobre la línea en memoria.
- El fichero tiene que estar guardado en forma ASCII. Si se intenta juntar un fichero guardado en forma binaria, se produce un error "Bad File Mode" (estado incorrecto del fichero).
- Especificando R, el programa juntado se ejecuta desde el principio, en cuanto termina el MERGE. Los ficheros abiertos no se cierran en este caso.

(Ejemplo)

MERGE "0:EX3"

Si el programa

10 A=123

20 B=456

30 S=A+B

50 PRINT S

se coloca en memoria, y el contenido del fichero EX3 en el disquete flexible 0 es:

```
20 B=789
40 S1=A-B
50 PRINT S,S1
60 END
```

el comando MERGE hace que el programa en memoria sea:

```
10 A=123
20 B=789
30 S=A+ B
40 S1=A-B
50 PRINT S,S1
60 END
```


3.1.18. SKIPF (saltar fichero)

(Función)

El comando SKIPF hace avanzar la cinta de cassette hasta el fichero situado a continuación del especificado.

(Formato)

SKIPF["[CAS0:]nombre de fichero"]

(Abreviado SK.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El comando SKIPF verifica las sumas de prueba (checksum) en los ficheros durante el salto de fichero. Si el resultado de la suma de prueba es incorrecto, se produce un error "Device I/O Error" (Error de dispositivo de entrada/salida).
- Si se omite el descriptor de ficheros, la cinta avanza hasta la posición inmediatamente a continuación del primer fichero.

(Ejemplo)

SKIPF "DEMO"

La cinta avanza a la posición que sigue a un fichero llamado DEMO.

3.1.19. DSKINI (inicializar disco)

(Función)

El comando DSKINI inicializa el directorio de un disquete flexible.

(Formato)

DSKINI número de unidad de discos

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se especifica un número de unidad de disco entre 0 y 3 para indicar la unidad que contiene el disquete que se va a inicializar.
- Cuando este comando se introduce en el modo directo, se visualiza la pregunta "Are you sure (Y or N)? (Está Vd. seguro? (Sí o No)). Como respuesta se introduce Y para inicializar, o N para anular este comando.

(Ejemplo)

FILES "1:"

WRITE 0 B S 1
SALES 0 A S 6

145 Clusters Free (145 grupos libres)

DSKINI 1

Are you sure (Y or N)? Y

Ready

FILES "1:"

152 Clusters Free (152 grupos libres)

Ready

3.1.20. EXEC (ejecutar)

(Función)

El comando EXEC ejecuta programas en lenguaje máquina.

(Formato)

EXEC[dirección de comienzo]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El comando EXEC ejecuta un programa en lenguaje máquina que está cargado en memoria, empezando en la dirección especificada por el operando. Si se omite este operando, la ejecución empieza en la dirección de entrada del último comando LOADM que se ha ejecutado, o en la dirección de comienzo especificada en el comando EXEC anterior.
- Después de la ejecución de una subrutina en lenguaje máquina por una sentencia EXEC, el comando en lenguaje máquina RTS devuelve el control a la sentencia que sigue a EXEC.

(Ejemplos)

EXEC &H5000

Se ejecuta un programa en lenguaje máquina empezando en la dirección &H5000.

10 EXEC &H5000
20 ...

Suponiendo que una subrutina en lenguaje máquina está almacenada en la dirección &H5000, se ejecuta esta subrutina, y entonces el control vuelve a la línea 20.

3.1.21. LOADM (cargar programa máquina)

(Función)

El comando LOADM carga un programa máquina en memoria y lo ejecuta.

(Formato)

LOADM "descriptor de ficheros" [, [desplazamiento] [, R]]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El programa que se va a cargar debe estar almacenado en un fichero en lenguaje máquina creado por el comando SAVEM.
- Cuando se especifica un valor de desplazamiento, el fichero se carga en la dirección obtenida al sumar el desplazamiento y la dirección de comienzo especificada por el comando SAVEM que había creado el fichero. Por lo tanto, el programa en lenguaje máquina debe estar escrito en un código independiente de la posición. Si se omite el desplazamiento, el fichero se carga en la misma dirección de comienzo especificada cuando se salvó.

Cuando se carga un programa en lenguaje máquina mediante LOADM, siempre hay que reservar espacio de memoria para ello, usando una sentencia CLEAR para limitar las direcciones a las que accede el BASIC.

Si se especifica R, la ejecución empieza en la dirección de entrada tan pronto como el programa se cargue. Si se omite R, una vez cargado el programa, el BASIC vuelve al nivel de comandos.

(Ejemplos)

LOADM "CAS0: SUB1",&H500

Un programa con nombre de fichero SUB1 se carga desde una cinta cassette a la memoria, y se ejecuta. Se suma un desplazamiento &H500 a la dirección base para calcular la dirección de carga.

LOADM"0:SUB2",,R

Un programa con nombre de fichero SUB2 se carga desde el disco flexible 0 a la memoria y se ejecuta.

3.1.22. SAVEM (guardar programa máquina)

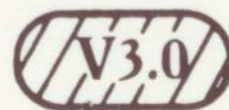
(Función)

El comando SAVEM almacena el contenido de memoria en un fichero especificado.

(Formato)

SAVEM "descriptor de fichero", dirección de comienzo, dirección final, dirección de entrada

(Versiones)



(Descripción)

El comando SAVEM almacena en un fichero especificado un programa en lenguaje máquina existente en la memoria.

Se puede definir la zona de memoria a almacenar indicando las direcciones de comienzo y final en la memoria.

La dirección de entrada identifica el punto de entrada del programa; es decir, el punto en el que se inicia la ejecución cuando el programa se carga en memoria mediante el comando LOADM.

Si el descriptor de ficheros especifica un disco flexible, el nombre del fichero debe ser uno nuevo. Si se especifica el nombre de un fichero que ya está en el disco, el BASIC hace lo siguiente. En el modo directo, se visualiza la pregunta "Are You sure (Y or N)? (Está Vd. seguro? (Sí o No)). Si se contesta con Y, entonces se elimina el fichero existente, y el programa se guarda en un fichero nuevo. Si se contesta con N, el fichero existente se mantiene en el disco, y no se guarda el programa. En el modo indirecto, el BASIC visualiza el mensaje "File Already Exist In line number" (fichero ya existe en el número de línea) y vuelve al nivel de comandos.

(Ejemplos)

10 SAVEM“CAS0: OBJ”,&H5000,&H5100,&H5000

Un programa en lenguaje máquina existente en la memoria entre las direcciones &H5000 a &H5100 se guarda en una cinta cassette y se le asigna el nombre de fichero OBJ. Su dirección de ejecución es &H5000.

3.1.23. HARDC (copia impresa)

(Función)

El comando HARDC envía a la impresora los datos de pantalla.

(Formato)

HARDC [{0|1|2}]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Existen tres opciones para el comando HARDC: 0, 1, y 2. La opción por omisión es 0.

- (1) 0: Se imprimen sólo los caracteres visualizados en pantalla.
- (2) 1: Cada punto en pantalla se imprime como cuatro puntos horizontales con una de estas tres intensidades: negro, gris o blanco.

Estas intensidades corresponden a los colores en pantalla como se indica a continuación:

Los puntos azules, rojos, verdes y blancos en pantalla se imprimen en negro.

Los puntos morados, azul claro y amarillos en pantalla se imprimen en gris.

Los puntos que aparecen en negro en pantalla no se imprimen (blanco).

- (3) 2: La impresora imprime cada punto no negro en pantalla como un punto.

(Ejemplo)

HARDC 2

La impresora imprime cada punto en pantalla como un punto, y los puntos que aparecen en negro en pantalla se omiten.

3.1.24. MON (monitor)

(Función)

El comando MON se utiliza para pasar del nivel de comandos en BASIC al nivel de comandos del monitor.

(Formato)

MON

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se pulsa la tecla BREAK (la tecla STOP para la V1.0 y V2.0) para volver al nivel de comandos en BASIC. CTRL-C y CTRL-X tendrán el mismo efecto que la tecla BREAK.
- Cuando se introduce el nivel de comandos del monitor, se visualiza el símbolo de "preparado" (*), que le permite utilizar los siguientes subcomandos.

Subcomando	Función
M	Modifica contenido de memoria.
G	Hace que el microprocesador comience a ejecutar programas a partir de una dirección especificada.
R	Visualiza el contenido de los registros y permite su modificación.
D	Visualiza el contenido de 64 bytes de datos empezando en una dirección especificada.

(1) M

(Formato)

M[dirección]

(Descripción)

- Cuando se introduce M y una dirección hexadecimal de 4 dígitos, el subcomando M visualiza el contenido de esa dirección y espera una entrada. Para cambiar el contenido, se introduce un nuevo número hexadecimal de uno o dos dígitos, después de la visualización. Para dejar el contenido tal como está, se pulsa RETURN. Estas operaciones pueden repetirse en la siguiente dirección. Si se omite la dirección, se visualiza el contenido de la última dirección utilizada. Para volver al nivel de comandos del monitor, basta con introducir un carácter no hexadecimal como dirección.

(Ejemplo)

```
MON
*M5000
5000 00-20
5001 00-09
5002 00-17
5003 00-X
*
```


(2) G

(Formato)

G[dirección a saltar]

(Descripción)

- Se especifica una dirección en hexadecimal de hasta cuatro dígitos. Si se omite este operando, el control se desvía según el contenido del registro PC establecido mediante el subcomando R.

(Ejemplo)

MON

*G5130

Este subcomando ejecuta un programa en lenguaje máquina, comenzando en la dirección &H5130.

(3) R

(Formato)

R

(Descripción)

- El subcomando R visualiza el contenido de los registros del microprocesador y los actualiza con los datos que el operador introduzca por el teclado. Los registros se visualizan en orden de CC, A, B, DP, X, Y, U, y PC, y su contenido puede actualizarse de la misma manera que con el subcomando M.

(Ejemplo)

```
MON
*R
CC B4-
A 00
B 44-
DP 00-
X AFA2-5649
Y C830-6517
U 0340-
PC AFA7-
```

*

(4) D

(Formato)

D[dirección]

(Descripción)

- El subcomando D visualiza 64 bytes de datos empezando en la dirección especificada. Si se omite la dirección, la visualización de los 64 bytes empieza a partir de la dirección que sigue a la última dirección visualizada.

(Ejemplo)

MON

*D9000

9000	E1	96	E1	9C	D9	A1	CD	BE
9008	D0	14	D1	F5	D1	ED	CA	A5
9010	D1	6C	D1	37	CE	8B	D6	BD
9018	DD	90	E6	24	DD	2E	CB	03
9020	ED	56	AB	74	9E	BD	9F	OD
9028	94	AE	CF	D7	CA	B1	BF	4D
9030	B0	F5	B1	23	93	FB	E9	53
9038	E9	6E	E6	B6	DE	E3	DF	91

*

3.1.25. TERM (terminal)

(Función)

El comando TERM pone al FM-7 en modo terminal.

(Formato)

TERM["cbpsma"]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

El comando TERM permite usar el FM-7 como el terminal de otro ordenador usando el puerto de comunicación RS-232C. Se pueden fijar las siguientes especificaciones mediante el operando:

- c Especifica el reloj.

S: Reloj lento (1/64 frecuencia-división)

F: Reloj rápido (1/16 frecuencia-división)

Si se especifica S, se puede elegir entre una velocidad en baudios de 300, 600, 1200, 2400, ó 4800.

Si se especifica F, se puede elegir entre 1200, 2400, 4800, ó 9600. La velocidad en baudios se selecciona mediante los microinterruptores.

- b Especifica el número de bits de los datos

8: 8 bits

7: 7 bits

Si se selecciona 7, se dispone automáticamente de los códigos shift-in (SI) y shift-out (SO) para distinguir los códigos alfanuméricos de los códigos kana.

- p Especifica paridad par, impar o sin paridad.

N: no se verifica paridad

O: se verifica paridad impar

E: se verifica paridad par

- s Especifica el número de bits de parada.

2: 2 bits de parada.

1: 1 bit de parada.

- m Especifica el modo de comunicación.

F: Full duplex

H: Half duplex

- a Especifica si hay que realizar el salto de líneas automático.

A: Se realiza el salto de líneas automático.

N: No se realiza el salto de líneas automático.

Si se selecciona A, cada vez que se envía un código de retorno de carro (CR), se agrega automáticamente a continuación un código LF.

Si se omite el operando, se asume TERM "S8N2FN".

Para cambiar del modo terminal al modo de comandos en BASIC, pulsar la tecla BREAK (La tecla STOP para V1.0 y V2.0).

En el modo terminal, las teclas de función programable (PF) hacen lo siguiente:

PF6: Conecta / desconecta la función de eco a la impresora.

PF7: Full duplex o half duplex.

PF8: Activa/desactiva la visualización de los códigos de control &H00 a &H1F como caracteres.

PF9: Envía caracteres de visualización a la impresora.

PF10: Envía una señal de BREAK (aprox. 100 milisegundos).

PF6 a PF8 son teclas de cambio de estado: la función se invierte (cambia de activada a desactivada, y viceversa) cada vez que se pulsa la tecla. Estas tres teclas están activadas como se describe a continuación, al entrar en el modo terminal:

PF6: No realiza eco a la impresora.

PF7: Full duplex

PF8: No visualiza los códigos de control.

(Ejemplo)

TERM "S8N2FA"

Este comando TERM establece el puerto RS-232C en:

- Reloj lento. (la velocidad en baudios puede establecerse en 300, 600, 1200, 2400 ó 4800, con los microconmutadores.
- Longitud de datos de 8 bits, dos bits de paro, sin bit de paridad.
- Salto de líneas automático al recibirse el código CR.

3.1.26. EDIT (editar)

(Función)

La sentencia EDIT visualiza en pantalla una línea especificada.

(Formato)

EDIT número de línea

(Versiones)

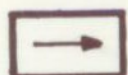
V1.0

V2.0

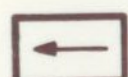
V3.0

(Descripción)

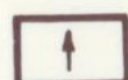
- Cuando se ejecuta una sentencia EDIT, se borra la pantalla. A continuación se visualiza la línea especificada con el cursor situado en la posición que sigue de inmediato al número de línea. En ese momento, puede utilizar las siguientes teclas de edición y movimiento del cursor para editar la línea.



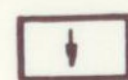
Desplaza el cursor a la derecha.



Desplaza el cursor a la izquierda.



Desplaza el cursor hacia arriba.



Desplaza el cursor hacia abajo.



Los caracteres se insertan en la posición indicada por el cursor. Pulsando la tecla **INS** se ilumina el piloto y se permite entrada de datos. Cuando se introduce un carácter, todos los caracteres posteriores empezando por la posición del cursor, se desplazan un espacio a la derecha, para hacerles sitio.

DEL

Elimina el carácter en la posición del cursor, y todos los caracteres que se encuentran a la derecha del cursor se desplazan un espacio a la izquierda, para cerrar el hueco.

EL

Elimina todos los caracteres de la línea a la derecha del cursor.

←

(**BS** para **V1.0** y **V2.0**) Elimina el carácter a la izquierda del cursor. Todos los caracteres siguientes se desplazan un espacio a la izquierda.

RETURN

Termina la edición y restaura el BASIC al nivel de comandos.

(Ejemplo) :

EDIT 300

3.2. Sentencias Generales

3.2.1. DEF FN (definir función)

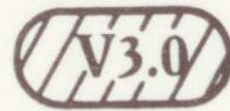
(Función)

La sentencia DEF FN define y nombra una función del usuario.

(Formato)

DEF FN nombre[(lista de parámetros)]=expresión

(Versiones)



(Descripción)

- El nombre ha de ser un nombre de variable numérica o de variable de cadena, precedido de FN para formar un nombre de función.
- La lista de parámetros es una lista de nombres de variables que son sustituidos por valores reales cuando se llama a la función. Los nombres de variables han de ir delimitados por comas (,). Cuando se llama a la función, hay que especificar una expresión (parámetro real) para cada variable en la lista de parámetros.
- El tipo de nombre de función ha de coincidir con el tipo de resultado de la expresión (numérico o de cadena). El tipo de cada nombre de variable en la lista de parámetros también ha de coincidir con el tipo del parámetro real correspondiente.
- La sentencia DEF FN ha de ser codificada antes de llamar a la función a la que define. Esta sentencia no se puede usar en el modo directo.
- Un nombre de variable en una lista de parámetros puede ser utilizado en otras líneas del mismo programa. El BASIC las trata como variables distintas.

(Ejemplo)

```
10 DEF FNP(X)=3.14*X*X
20 AREA=FNP(R1)
```


3.2.2. DEF USR (definir usuario)

(Función)

La sentencia DEF USR especifica la dirección de comienzo de un programa en lenguaje máquina.

(Formato)

DEF USR[numeral]=entero

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El numeral puede ser cualquier dígito de 0 a 9. Le precede USR para formar el nombre de función, por el cual se llama a la función. Si se omite el numeral, se asume 0.
- El entero especifica la dirección de comienzo de ejecución de un programa en lenguaje máquina que se encuentra en memoria. El programa en lenguaje máquina debe estar almacenado entre la dirección especificada por el segundo operando del comando CLEAR y la dirección más baja del código del sistema F-BASIC.
- Se llama a un programa en lenguaje máquina que ha sido nombrado por la función, de la misma manera en que se llama a cualquier función. El formato es:

USR[número](argumento)

El argumento debe ser una cadena o una expresión numérica. La información dada por este argumento se pasa a la subrutina en lenguaje máquina usando el acumulador y el registro X.

Acumulador =

Tipo de argumento

2 Entero

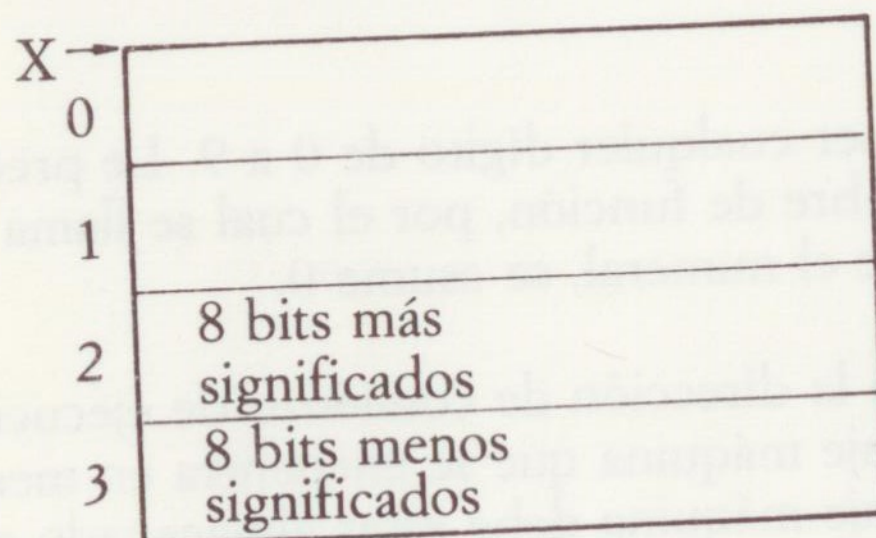
3 Cadena

4 Real de simple precisión

8 Real de doble precisión

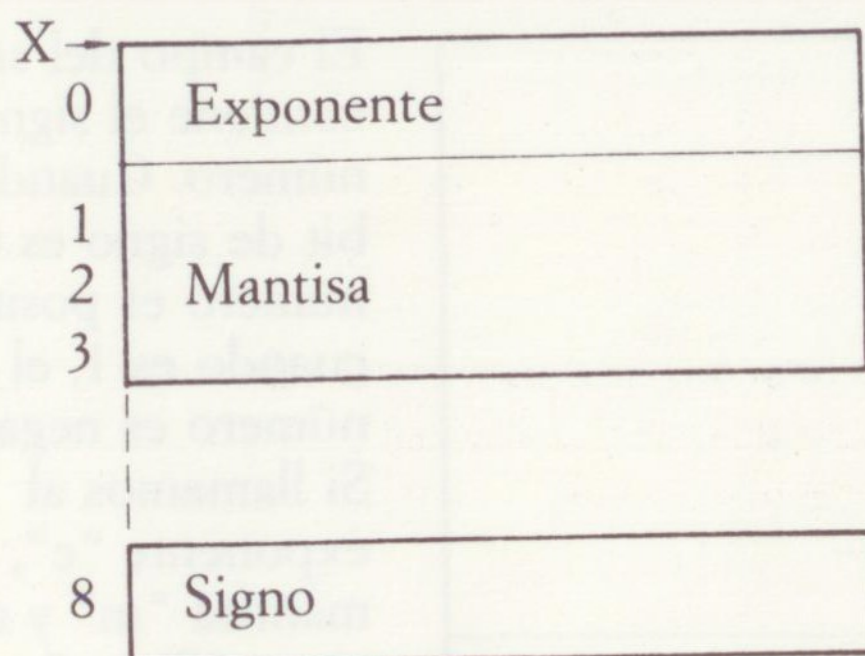
El registro X proporciona la dirección en la cual se almacena el valor del argumento. El formato de almacenamiento es el siguiente: (Para variables numéricas, los argumentos se transfieren a direcciones fijas en el área de trabajo):

(1) Argumento entero



El valor en complemento a dos se coloca en las direcciones 2 y 3 a partir de la dirección indicada por el registro X.

(2) Argumento real de simple precisión



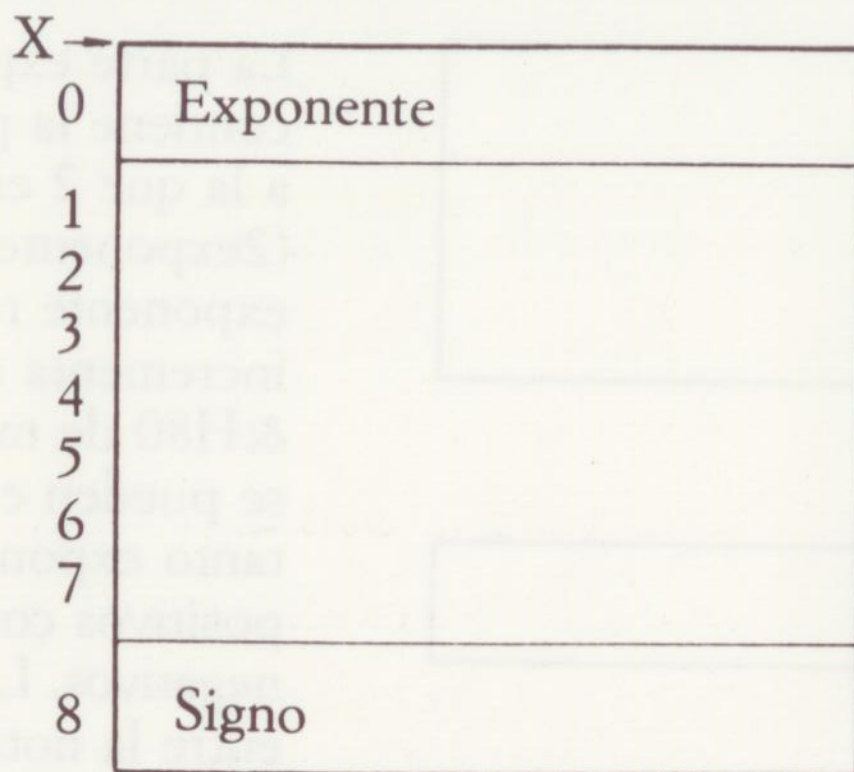
La parte exponente contiene la potencia a la que 2 es elevada ($2^{\text{exponente}}$). El exponente real se incrementa con &H80 de modo que se pueden expresar tanto exponentes positivos como negativos. La relación entre la notación en exceso &H80 y los valores reales es la siguiente:

De &H00 a &H80
de 2^{-128} a 2^0

De &H80 a &HFF
de 2^0 a 2^{127}

La parte de mantisa contiene un número binario sin signo.

(3) Argumento real de doble precisión

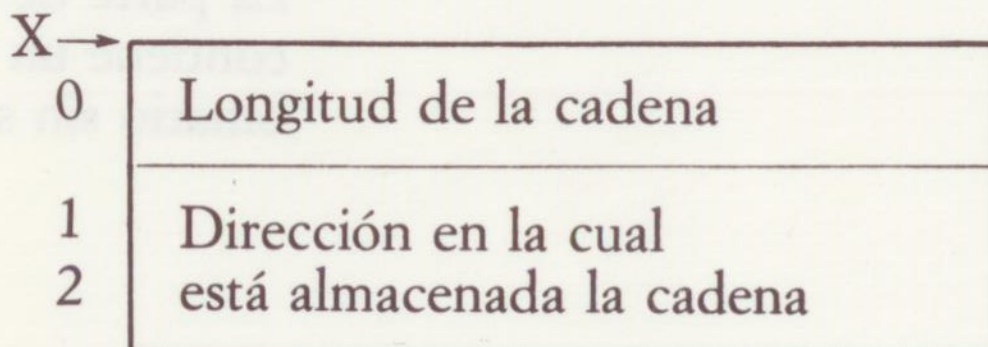


El campo del signo contiene el signo del número. Cuando el bit de signo es 0, el número es positivo, y cuando es 1, el número es negativo. Si llamamos al exponente "e", a la mantisa "m" y al signo "Signo", el valor del argumento X se expresa como:

$$X = \text{Sign} \cdot 0.m \times 2^e$$

(4) Argumento cadena de caracteres

El registro X almacena la dirección de cabecera de tres bytes de datos llamados descriptor de cadena.



(Ejemplo)

```
10 A%=56
20 DEF USRO=&H6100
30 B%=USRO(A%)
```

La ejecución de USRO coloca un 2 en el registro A y en el registro X se almacena la dirección de memoria en la cual está contenido 56. Después ejecuta el programa en lenguaje máquina almacenado en la dirección &H6100.

3.2.3. DEFINT/SNG/DBL/STR (definir simple/doble/cadena)

(Función)

La sentencia DEF declara si una variable es de tipo entero, simple precisión, doble precisión o cadena.

(Formato)

DEF especificación de tipo gama de caracteres

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La sentencia DEF declara que todos los nombres de variables que comiencen con un determinado carácter dentro de una gama especificada, son del tipo indicado.
- Se pueden declarar cuatro tipos de variables.

DEFINT	Declara tipo entero.
DEFSNG	Declara tipo de simple precisión.
DEFDBL	Declara tipo de doble precisión.
DEFSTR	Declara tipo cadena.

- La gama de caracteres se especifica con caracteres alfabéticos. Se utilizan comas para separar los caracteres, y guiones para designar gamas de caracteres.
- Un carácter de declaración de tipo conectado a una variable tiene prioridad sobre la declaración con la sentencia DEF.
- Las variables cuyo tipo no ha sido especificado mediante una sentencia ni mediante un carácter, son tratadas como variables de simple precisión.

(Ejemplos)

DEFINT A,C,E

Esta sentencia declara que todas las variables que comiencen por A, C, y E son enteras.

DEFSNG B-X

Esta sentencia declara que todas las variables que comiencen con letras entre la B y la X son de simple precisión.

DEFDBL D-G

Esta sentencia declara que todas las variables que comiencen con D, E, F, y G son de doble precisión.

DEFSTR A,E-G

A%=25

Esta sentencia declara que todas las variables que comiencen con A, E, F, y G son cadenas. A% es una variable entera debido al carácter de declaración de tipo que tiene prioridad sobre la sentencia de declaración de tipo.

3.2.4. REM (comentario)

(Función)

La sentencia REM se usa para incluir comentarios en un programa.

(Formato)

REM cadena de caracteres comentario

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La sentencia REM es una sentencia no ejecutable que aparece como comentario del programa.
- Se puede introducir una comilla simple (') en lugar de la palabra clave REM.
- El signo dos puntos (:) se considera también como parte del comentario en la sentencia REM. Esto quiere decir que no se puede terminar un comentario con el signo de dos puntos y continuar con la siguiente sentencia.

(Ejemplo)

```
10 REM PROGRAMA No. 10  
20 'FECHA 4, 10, 1981
```


3.2.5. END (fin)

(Función)

La sentencia END termina la ejecución del programa, cierra todos los ficheros abiertos, y hace que el BASIC vuelva al nivel de comandos.

(Formato)

END

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Una sentencia END se puede colocar en cualquier punto del programa para detener su ejecución.
- Un programa no tiene por qué terminarse con una sentencia END, pero si no se especifica esta sentencia los ficheros abiertos no se cierran.

(Ejemplo)

```
10 A=123
20 B=456
30 C=A+B
40 PRINT C
50 END
```


3.2.6. FOR NEXT (para siguiente)

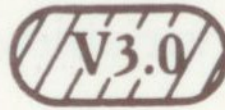
(Función)

La sentencia FOR NEXT ejecuta repetitivamente una serie de sentencias.

(Formato)

FOR variable=expresión 1 TO expresión 2 [STEP
expresión 3]

(Versiones)



(Descripción)

- La variable es llamada variable de control, y sirve como contador del bucle. Debe ser una variable numérica de un tipo que no sea doble precisión.
- Expresión 1, expresión 2 y expresión 3 son expresiones numéricas que especifican el valor inicial, el valor final, y el paso. Si se omite el paso (STEP), se toma 1.

- Una sentencia FOR NEXT funciona del siguiente modo:

Cuando se encuentra una sentencia FOR, la variable toma el valor de la expresión 1, y se ejecutan una vez las sentencias comprendidas entre el par FOR y NEXT. Cuando se encuentra el NEXT, el valor de la variable se incrementa en el paso indicado por la expresión 3, y se compara con el valor final de la expresión 2. Si es mayor que el valor final, la ejecución pasa a la sentencia que sigue al NEXT. Si es menor o igual al valor final, el control vuelve a la sentencia que sigue a FOR y se repite el mismo proceso. Esto se llama bucle FOR NEXT.

El bucle anterior usa un valor positivo. Si el paso (STEP) es negativo, el valor de la expresión 1 debe ser mayor que el de la expresión 2, y el bucle FOR NEXT continúa hasta que el valor del contador es menor que la expresión 2.

- Si el paso es positivo y el valor inicial es mayor que el final, o si el paso es negativo y el valor inicial es menor que el final, el bucle FOR NEXT se ejecuta una sola vez.
- No se debe cambiar el valor de las expresiones 1, 2 y 3 durante la ejecución de un bucle FOR NEXT.

Bucles anidados

Pueden especificarse bucles FOR NEXT anidados es decir, se puede incluir un bucle FOR NEXT dentro de otro bucle FOR NEXT. En este caso, el bucle interior debe estar completamente contenido en el bucle exterior. Al codificar bucles anidados, debe usarse una variable de control para cada bucle FOR NEXT.

(Ejemplos)

```
10 PI=3.14159
20 FOR I=0 TO 360 STEP 10
30 S=SIN(PI/180*I)
40 PRINT S
50 NEXT I
60 END
```

Este programa imprime una tabla de senos desde 0 hasta 360 grados en incrementos de 10 grados.

```
10 M=&H20
20 FOR I=&HFF TO M STEP -1
30 PRINT CHR$(I);
40 NEXT I
50 END
```

Este programa, con un valor de paso negativo, repite el bucle hasta que la variable de control I es menor que el valor final M.

```
10 FOR I=1 TO 9
20 FOR J=1 TO 9
30 PRINT I;"*";J;"=";J*I
40 NEXT J
50 PRINT
70 NEXT I
80 END
```

Este programa imprime una tabla de multiplicar desde 1 * 1 hasta 9 * 9. Ya que se anidan los bucles FOR NEXT; "I" se incrementa en 1 cada vez que la línea 30 se repite 9 veces.

3.2.7 NEXT (siguiente)

(Función)

La sentencia NEXT marca el final de un bucle FOR NEXT.

(Formato)

NEXT[nombre de variable[,nombre de variable]...]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Una sentencia NEXT corresponde a la última sentencia FOR que le precede y no esté emparejada con otro NEXT.
- El nombre de variable debe concordar con el nombre de la variable de control de la correspondiente sentencia FOR.
- Se puede utilizar una sola sentencia NEXT para terminar más de un bucle FOR NEXT escribiendo los nombres de las variables de control en el orden en el cual deben terminarse las sentencias FOR.

(Ejemplo)

Ver Sección 3.2.6.

3.2.8 GOTO (ir a)

(Función)

Una sentencia GOTO bifurca incondicionalmente al número de línea especificado.

(Formato)

GOTO número de línea

(Abreviado GO.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Si la línea indicada en el número de línea es una sentencia ejecutable, la ejecución continúa a partir de ese punto. Si se trata de una sentencia no ejecutable, el control pasa a la primera sentencia ejecutable que encuentre a partir de esa línea.

(Ejemplo)

```
10 INPUT A,B
20 PRINT A+B
30 PRINT A-B
40 PRINT A * B
50 PRINT A/B
60 GOTO 10
70 END
```

Una sentencia GOTO también se puede especificar insertando un espacio entre GO y TO.
Por ejemplo: 60 GO TO 10.

El programa de este ejemplo suma, resta, multiplica y divide los datos de entrada, e imprime los resultados. Dado que la sentencia GOTO bifurca a la línea 10, el programa puede ser repetido tantas veces como se desee. El programa vuelve al nivel de comandos BASIC, cuando se pulsa la tecla BREAK (STOP para V1.0 y V2.0) CTRL-C o CTRL-X.

3.2.9 ON GOTO (en ir a)

(Función)

La sentencia ON GOTO bifurca, según el valor de la expresión, a una de entre varias líneas cuyos números se especifican.

(Formato)

ON expresión GOTO número de línea[,número de línea]...

(Versiones)



(Descripción)

- El control salta al primer número de línea si el valor de la expresión es 1, al segundo número de línea si el valor de la expresión es 2, y así sucesivamente. Si el valor de la expresión no es entero, se redondea al entero más próximo (se redondea hacia arriba si la fracción es .5 o mayor, y hacia abajo si es menor que .5).
- Si el valor de la expresión es 0 o es mayor que el número de líneas especificado, el control procede con la siguiente sentencia.
- Se produce un error "Illegal Function Call" (llamada ilegal a función) si el valor de la expresión es negativo.

(Ejemplo)

```
10 INPUT A
20 ON A GOTO 30,50
30 B=1
40 GOTO 60
50 B=2
60 FOR I=1 TO B
70 BEEP
80 PRINT "Beep"
90 FOR J=1 TO 130
100 NEXT J
110 NEXT I
```

Si el dato que se introduce es 1, el programa salta a la línea 30; si es 2, salta a la línea 50. Si el dato que se introduce es 1, la señal acústica suena una vez; si es 2, suena dos veces.

3.2.10. GOSUB (ir a subrutina)

(Función)

La sentencia GOSUB llama a una subrutina. Al final de la subrutina, el control vuelve a la sentencia que sigue a GOSUB.

(Formato)

GOSUB número de línea

(Abreviado GOS.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de línea especifica el número de la línea de comienzo de la subrutina a la que se va a llamar.

(Ejemplo)

```
10 N$="F-BASIC"  
20 GOSUB 100  
30 PRINT N$  
40 END  
100 PRINT N$  
110 N$="Ver 3.0"  
120 RETURN
```

En este programa, el control se bifurca en la sentencia GOSUB a la línea 100, y luego vuelve desde la sentencia RETURN a la línea 30.

3.2.11. RETURN (retorno)

(Función)

La sentencia RETURN termina una subrutina y devuelve el control al programa que hace la llamada.

(Formato)

RETURN[número de línea]

(Abreviado RET.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Una sentencia RETURN puede colocarse en cualquier sitio y cuantas veces se desee, en una subrutina.
- Si se especifica el número de línea, el control salta a la línea indicada. Si se omite este número, entonces el control vuelve a la sentencia que sigue al último GOSUB ejecutado. Ver la sentencia ON para detalles sobre lo que ocurre cuando se especifica la sentencia RETURN en una rutina de proceso de interrupción.

(Ejemplo)

Ver la Sección 3.2.10.

3.2.12. ON GOSUB (en ir a subrutina)

(Función)

La sentencia ON GOSUB llama a una subrutina que empieza en uno de los varios números de línea especificados, según el valor de una expresión.

(Formato)

ON expresión GOSUB número de línea[,número de línea]...

(Versiones)



(Descripción)

- El control salta al primer número de línea si el valor de la expresión es 1, salta al segundo número de línea si el valor de la expresión es 2, y así sucesivamente. Si el valor no es un entero, se le redondea al entero más próximo.
- Si el valor de la expresión es 0 o es mayor que el número de números de línea especificados, el control pasa a la siguiente sentencia. Si el valor es negativo, se produce un error de "Illegal Function Call" (llamada ilegal a función).
- El número de línea debe ser el primer número de línea de la subrutina que se va a llamar.

(Ejemplo)

```
10 S1=0
20 S2=0
30 INPUT A
40 ON A GOSUB 100,200,300
50 GOTO 30
60 END
100 "SUB 1
110 S1=S1+1
120 RETURN
200 S2=S2+1
210 RETURN
300 PRINT S1,S2
310 RETURN 60
```

Este programa entra en la subrutina en la línea 100 si el dato de entrada es 1, y entra en la subrutina en la línea 200 si el dato de entrada es 2. Después de llegar a una sentencia RETURN, el control vuelve a la línea 50 del programa principal. Si el dato de entrada es 3, se ejecuta la subrutina en la línea 300, y entonces el control vuelve a la línea 60 ya que la sentencia RETURN especifica la línea 60.

3.2.13. STOP (parar)

(Función)

La sentencia STOP detiene la ejecución del programa y restaura el BASIC al nivel de comandos.

(Formato)

STOP

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Una sentencia STOP puede aparecer en cualquier sitio de un programa para detener la ejecución.

Cuando se ejecuta la sentencia STOP, aparece el siguiente mensaje:

Break In nnnnn

nnnnn identifica el número de línea de la sentencia STOP.

- A diferencia de END, STOP no cierra los ficheros.
- La ejecución de una sentencia STOP restaura el BASIC al nivel de comandos. Después, si se desea reanudar la ejecución del programa en el punto en que fue interrumpida, se introduce un comando CONT.

(Ejemplo)

```
10 S=0:S1=0
20 FOR I=1 TO 100
30 S=S+1:S1=S1+I^2
40 NEXT
50 PRINT S
60 STOP
70 PRINT S1
80 END
```

La ejecución de la sentencia STOP en la línea 60 hace que vuelva el BASIC al nivel de comandos. Si se introduce un comando CONT, el programa empezará de nuevo a partir de la línea 70.

3.2.14. IF THEN ELSE (si, entonces)

(Función)

La sentencia IF THEN ELSE selecciona una sentencia que se va a ejecutar dependiendo del resultado de una expresión.

(Formato)

IF expresión THEN {sentencia | número de línea}
[ELSE {sentencia | número de línea}]

o

IF expresión GOTO número de línea [ELSE {sentencia
número de línea}]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser una expresión lógica o relacional.
- Si el valor de la expresión es cierto, (no 0), se ejecuta la siguiente sentencia THEN o GOTO. La sentencia THEN puede ir seguida de una sentencia que se vaya a ejecutar a continuación, o bien de un número de línea al que bifurcará el control. La sentencia GOTO debe ir seguida por el número de línea al que bifurcará el control.
- Si el valor de la expresión es falso, se pasa por alto la siguiente sentencia THEN o GOTO, y si una sentencia ELSE le sigue a la THEN o GOTO, es la siguiente en ejecutarse. Si no hay una sentencia ELSE, el control pasa a la siguiente sentencia.
- Anidado de sentencias IF

Una sentencia IF puede tener otra sentencia IF que sigue a su sentencia THEN o ELSE. Si el número de sentencias THEN y el número de sentencias ELSE es distinto en una sentencia IF, cada ELSE corresponde a la THEN más próxima que todavía no haya sido asociada con otra ELSE. El número de niveles de anidado es ilimitado dentro de una línea.

(Ejemplo)

```
10 INPUT A,B
20 IF A>B THEN A=B
30 PRINT A
40 END
```

Se imprime el menor de los valores de entrada A o B.

Ready

RUN

? 3,7

3

Ready

```
10 INPUT A,B,C
20 IF A>B THEN IF A >C THEN
  PRINT A ELSE PRINT C
  ELSE IF B >C THEN PRINT B
  ELSE PRINT C
30 GO TO 10
```

Se imprime el mayor de los valores introducidos A, B o C.

Ready

RUN

? 9,6,20

20

? 2,5,8

8

?

3.2.15. WHILE WEND (mientras hasta)

(Función)

La sentencia WHILE WEND ejecuta repetidamente una serie de instrucciones, según la condición especificada.

(Formato)

WHILE expresión

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser una expresión lógica o relacional.
- Las líneas del programa entre una sentencia WHILE y la sentencia asociada WEND se llaman bucle WHILE WEND.
- El bucle WHILE WEND se ejecuta repetidamente mientras el valor de la expresión sea cierto (no 0). Cada vez que se va a repetir el bucle, se evalúa la expresión; si se encuentra que no es cierta, la ejecución pasa a la sentencia que sigue a la WEND. Si el valor de la expresión resultase no cierto en la primera prueba, entonces el bucle no se ejecutaría ni una sola vez.
- Los bucles WHILE WEND pueden anidarse en tantos niveles como permita la memoria disponible. Cada sentencia WEND se asocia con la sentencia WHILE ejecutada recientemente. Si no se emparejan correctamente las sentencias WEND y WHILE, se produce un error "While Wend" o "Wend Without While" (Wend sin While).

(Ejemplo)

```
10 WHILE I < 90
20 I=RND(1)*100
30 PRINT I
40 WEND
50 END
```

El bucle WHILE WEND se repite mientras que la variable I sea menor que 90.

Ready
RUN

59.1065
20.7991
55.0967
63.5863
10.411
80.6334
4.29073
95.3862

Ready

3.2.16. WEND (fin de mientras)

(Función)

La sentencia WEND marca el final de un bucle WHILE WEND.

(Formato)

WEND

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Una sentencia WEND se asocia con la última WHILE que le precede que no esté emparejada con otra WEND.

(Ejemplo)

Ver la Sección 3.2.15.

3.2.17. LET (hacer)

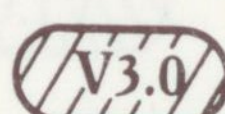
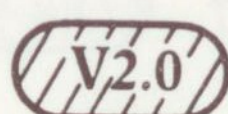
(Función)

La sentencia LET asigna a una variable el resultado de una expresión.

(Formato)

[LET] nombre de variable expresión

(Versiones)



(Descripción)

- La palabra clave LET es optativa.
- Si la expresión especificada es numérica, la variable ha de ser numérica; si la expresión es de tipo cadena, la variable indicada ha de ser de tipo cadena. Si no se cumplen estas reglas se genera un error "Type Mismatch" (Tipos no coincidentes).

(Ejemplo)

```
10 LET A=5  
20 LET B$="FUJITSU"
```

Estas sentencias LET asignan el valor 5 a la variable A y la cadena "FUJITSU" a la variable B\$. También pueden escribirse del siguiente modo:

```
10 A=5  
20 B$="FUJITSU"
```


3.2.18. SWAP (intercambiar)

(Función)

La sentencia SWAP intercambia los valores de dos variables.

(Formato)

SWAP variable,variable

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Las variables especificadas pueden ser enteras de simple precisión, de doble precisión, o cadenas, pero han de ser ambas del mismo tipo. Si no lo son, se produce un error "Type Mismatch" (Tipos no coincidentes).
- Si la segunda variable es una variable simple (es decir, no es una matriz), debe tener un valor previamente asignado.

(Ejemplo)

```
10 INPUT A,B
20 PRINT "A=";A,"B=";B
30 SWAP A,B
40 PRINT "A=";A,"B=";B
50 END
```

La ejecución de SWAP en la línea 30 intercambia los valores de las variables A y B.

Ready
RUN

2,5

A=2

B=5

A=5

B=2

Ready

3.2.19. DIM (dimensión)

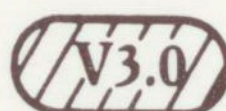
(Función)

La sentencia DIM especifica las dimensiones de una matriz, y reserva espacio en memoria para la matriz.

(Formato)

DIM nombre de variable(valor máximo del subíndice[,máximo valor del subíndice]...)

(Versiones)



(Descripción)

- El número de subíndices escritos entre paréntesis indica el número de dimensiones de la matriz.
- La sentencia DIM inicializa todos los elementos de la matriz a 0.
- Si se usa un nombre de variable matriz antes de haberse declarado por una DIM, se declara implícitamente con un valor máximo de subíndice de 10 por cada dimensión.
- Cuando se hace referencia a una variable matriz, el valor del subíndice en cada dimensión debe ser igual o menor que el valor máximo del subíndice declarado por DIM, y el número de dimensiones debe ser igual al declarado por DIM. De lo contrario, se produce un error "Subscript Out of Range" (Subíndice fuera del rango).
- Se puede dimensionar más de una matriz a la vez, utilizando comas para separarlas.

(Ejemplo)

10 DIM A(1,20)

Se declara una matriz con dos dimensiones.

20 DIM L%(10)

Se declara una matriz tipo entero.

30 DIM B\$(40)

Se declara una matriz cadena.

40 DIM C(40,20), D(50), E\$(26)

Se dimensionan tres matrices con una sola sentencia DIM.

3.2.20. POKE (poner)

(Función)

La sentencia POKE escribe datos en una dirección de memoria especificada.

(Formato)

POKE dirección especificada,dato

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La dirección especificada debe estar entre 0 y 65535 para números decimales, y entre &H0 y &HFFFF para números hexadecimales.
- Los datos deben estar comprendidos entre 0 y 255. Para los datos hexadecimales, la gama varía entre &H0 y &H100.

(Ejemplo)

10 POKE &H5000,57

Esta sentencia POKE escribe el decimal 57 en la dirección hexadecimal &H5000.

3.2.21. DATA (datos)

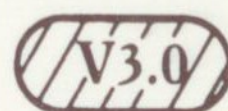
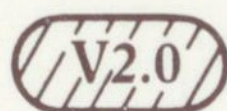
(Función)

La sentencia DATA almacena constantes numéricas y/o cadena que pueden leerse mediante las sentencias READ.

(Formato)

DATA constante[,constante]...

(Versiones)



(Descripción)

- Una sentencia DATA es una sentencia no ejecutable que puede ser insertada en cualquier punto del programa.
- Las constantes se delimitan por comas, y pueden escribirse hasta alcanzar la longitud máxima de la línea. Pueden usarse tantas sentencias DATA como sean necesarias. Las constantes definidas por las sentencias DATA son interpretadas como una serie de elementos de datos consecutivos, comenzando por la sentencia DATA de número más bajo.
- Cada constante puede ser una constante numérica (entera, de coma fija, de punto flotante, hexadecimal, u octal) o una constante cadena. Una constante cadena que contenga una coma (,) o un signo de dos puntos (:), o que esté precedida y/o seguida de espacios, ha de especificarse entre comillas ("). Las demás constantes no precisan comillas.
- Las constantes definidas por las sentencias DATA se leen en el orden en que se han especificado. Hay que asegurarse que la constante y la variable a la que se asigna son del mismo tipo.
- Una sentencia RESTORE permite volver a usar las constantes de las sentencias DATA. La lectura empieza a partir de la primera sentencia DATA.

(Ejemplo)

```
10 READ A,B
20 IF A=0 THEN END
30 C=A+B
40 PRINT "A+B=";C
50 GOTO 10
60 DATA 10,20,23,5
70 DATA 30,24,0,0
```

Ready

RUN

A+B = 30

A+B = 28

A+B = 54

Ready

3.2.22. READ (leer)

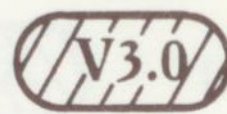
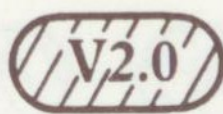
(Función)

La sentencia READ lee constantes definidas por sentencias DATA y las asigna a variables.

(Formato)

READ nombre de variable[,nombre de variable]...

(Versiones)



(Descripción)

- Las sentencias READ se usan siempre en combinación con las sentencias DATA. Cuando se ejecuta una sentencia READ, las constantes de las sentencias DATA se leen en orden y se asignan a las variables en la sentencia READ.
- Las variables de la sentencia READ pueden ser numéricas o cadena, pero cada una debe ser del mismo tipo que el de la constante correspondiente en la sentencia DATA.
- Si el número de variables especificado en las sentencias READ de un programa es mayor que el número de constantes indicadas en las sentencias DATA, se produce un error "Out of Data" (fin de datos).
- Ejecutando una sentencia RESTORE, las constantes que ya se han leído pueden volver a usarse, empezando a partir de la primera sentencia DATA, a no ser que se especifique de otro modo.

(Ejemplo)

Ver la Sección 3.2.21.

3.2.23. RESTORE (restaurar)

(Función)

La sentencia RESTORE hace que la lectura empiece a partir de la primera sentencia DATA.

(Formato)

RESTORE[número de línea]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Cuando se ejecuta una sentencia RESTORE, la siguiente sentencia READ empieza a leer datos a partir de la primera sentencia DATA del programa. Si se especifica el número de línea, la lectura comienza con la primera sentencia DATA que aparezca en/o a partir de dicha línea indicada.

(Ejemplo)

```
10 READ A,B,C
20 READ D,E,F
30 PRINT "A=";A;"B=";B;"C=";C
40 PRINT "D=";D;"E=";E;"F=";F
50 PRINT
60 RESTORE 130
70 READ A,B,C
80 RESTORE
90 READ D,E,F
100 PRINT "A=";A;"B=";B;"C=";C
110 PRINT "D=";D;"E=";E;"F=";F
120 DATA 1,3,5
130 DATA 2,4,6
```

Ready

Run

A=1 B=3 C=5

D=2 E=4 F=6

A=2 B=4 C=6

D=1 E=3 F=5

Ready

La sentencia RESTORE de la línea 60 hace que la sentencia READ de la línea 70 lea datos de la sentencia DATA de la línea 130. La RESTORE de la línea 80 hace que la READ de la línea 90 lea datos de la sentencia DATA de la línea 120.

3.2.24. LSET, RSET (ajustar a la izquierda/ajustar a la derecha)

(Función)

La sentencia LSET o RSET transfiere datos cadena a un buffer de ficheros de acceso directo.

(Formato)

{ LSET/RSET } variable de cadena = expresión

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Antes de la ejecución de una sentencia RSET o LSET, ha de haber sido ejecutada una sentencia FIELD.
- La variable cadena ha de ser un nombre de variable asignado mediante la sentencia FIELD.
- La expresión debe ser una expresión cadena, cuyo valor se va a asignar a la variable.
- Si la cadena resultante de la expresión es más corta que la longitud de campo definido por la sentencia FIELD, la sentencia LSET almacena los datos ajustados a la izquierda y la sentencia RSET almacena los datos ajustados a la derecha, y el espacio que queda se llena de espacios en blanco. Si la cadena es mayor que el campo los caracteres en exceso por la derecha se pierden.
- Para asignar un valor numérico a una variable cadena con la sentencia RSET o LSET, es necesario convertir el valor a formato cadena mediante la función MKI\$, MKS\$, o MKD\$.

(Ejemplo)

```
10 OPEN "R",#1,"TABLE"  
20 FIELD #1,20 AS A$,20 AS B$  
30 INPUT "DATA";C$,D$  
40 LSET A$=C$  
50 RSET B$=D$  
60 PUT #1,1  
70 CLOSE #1  
80 END
```


3.2.25. RANDOMIZE (aleatorizar)

(Función)

La sentencia RANDOMIZE altera una secuencia de números aleatorios.

(Formato)

RANDOMIZE [expresión]

(Abreviado RNDM.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica, y su valor debe estar entre -32768 y 32767.
- Si se omite la expresión, se detiene la ejecución del programa y aparece el siguiente mensaje:

Random Number Seed (-32768 to 32767)?

La ejecución del programa se reanuda introduciendo un número entre -32768 y 32767.

- Si no se cambia la serie de los números aleatorios, la función RND repite la misma secuencia de números aleatorios cada vez que el programa se ejecuta.

(Ejemplo)

```
10 RANDOMIZE
20 FOR I=1 TO 10
30 A=INT(RND(1)*8)+1
40 PRINT A;
50 NEXT
60 END
```

Este programa genera una serie de números aleatorios de 1 a 9. La sentencia RANDOMIZE de la línea 10 permite cambiar la serie de números aleatorios en cada ejecución.

Ready

RUN

Random Number Seed (-32768 to 32767)? 5649

2 3 5 1 1 8 3 5 2 3

Ready

3.2.26. ERROR (error)

(Función)

La sentencia ERROR simula toda ocurrencia de error en el BASIC y permite que el usuario defina los números de error.

(Formato)

ERROR número de error

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número del error debe ser un entero entre 1 y 255.
- La ejecución de una sentencia ERROR produce la aparición de un mensaje de error correspondiente al número indicado suspendiéndose la ejecución del programa si no hay ninguna sentencia ON ERROR GOTO. Si no existe ningún código de error coincidente con el número especificado, se visualiza el mensaje "Unprintable Error" (error no imprimible).
- Si hay una sentencia ON ERROR GOTO, el control salta al número especificado de línea sin que aparezca un mensaje de error.
- Cuando en cualquiera de los casos arriba citados se ejecuta una sentencia ERROR, el número del error y el número de línea donde está el error se asignan a las variables ERR y ERL.
- Con la sentencia ERROR, se pueden definir códigos de error del programa del usuario no reservados para el BASIC.

(Ejemplo)

```
10 ON ERROR GOTO 80
20 RANDOMIZE (TIME)
30 A=INT(RND(1)*9)+1
40 INPUT B
50 IF A<B THEN ERROR 80 ELSE IF A>B THEN ERROR 81
60 PRINT "Solución!!"
70 END
80 IF ERR=80 THEN PRINT "Demasiado grande!" ELSE PRINT
   "Demasiado pequeño!"
90 RESUME 40
100 END
```

Ready

RUN

? 5

Demasiado grande!

? 3

Solución !!

Ready

Este programa emplea una especie de juego de adivinar números. Si Vd. no acierta, el programa utiliza una función de detección de errores para procesar el "error". La rutina de proceso de errores determina la causa del error a partir del número de error definido con la sentencia ERROR.

3.2.27. ON ERROR GOTO (en error, ir a)

(Función)

La sentencia ON ERROR GOTO activa la función de detección de errores.

(Formato)

ON ERROR GOTO número de línea

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de línea indica el primer número de línea de la rutina de proceso de errores.
- Cuando se activa la función de detección de errores y ésta localiza un error, el programa transfiere el control a la rutina de proceso de errores especificada y no visualiza ningún mensaje.
- Si surge un error, las variables ERR y ERL toman automáticamente el valor del número del error y del número de línea.
- La función de detección no es válida dentro de una rutina de proceso de errores. Si se produce un error en una de tales rutinas, se visualiza un mensaje de error y la ejecución se para.
- Para desactivar la función de detección de errores se ejecuta ON ERROR GOTO 0.

(Ejemplo)

Ver la Sección 3.2.26.

3.2.28. RESUME (resumir)

(Función)

La sentencia RESUME restablece la ejecución de un programa después de procesar un error.

(Formato)

RESUME { NEXT | número de línea }

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Si se especifica el número de línea, el programa se reanuda en la línea indicada.
- Si se especifica NEXT, el programa sigue con la sentencia siguiente a aquella en la cual se había detenido el proceso.
- Si no se especifica ningún operando, el programa vuelve a la línea en la que se produjo el error. Obsérvese que el control volverá a la rutina de proceso del error si la causa del mismo no se ha eliminado.

(Ejemplo)

Ver la Sección 3.2.26.

3.2.29. BEEP (señal acústica)

(Función)

La sentencia BEEP hace que el altavoz integrado emita un tono.

(Formato)

BEEP [estado]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Especificando estado "1" la señal sigue sonando hasta que se ejecute un BEEP con estado "0".
- Si se omite el operando, el tono suena durante un breve intervalo, igual que cuando se ejecuta PRINT CHR\$(7).

(Ejemplo)

```
10 FOR I=1 TO 100
20 BEEP 0
30 FOR J=1 TO I
40 NEXT J
50 BEEP 1
60 FOR K=1 TO 15
70 NEXT K
80 NEXT I
90 BEEP 0
100 END
```

Se emiten tonos de longitud constante a intervalos de tiempo crecientes bajo el control de tres bucles FOR NEXT.

3.2.30. MOTOR (motor)

(Función)

La sentencia MOTOR controla el motor del grabador de cassette.

(Formato)

MOTOR [estado]

(Abreviado M.)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Especificando el estado ON el motor se pone en marcha, y especificando OFF se detiene.
- Si se omite el operando, el motor pasa de encendido a apagado o viceversa.

(Ejemplo)

MOTOR ON

El motor del cassette se pone en marcha.

3.2.31. TRON (activar trazas)

(Función)

La sentencia TRON traza la ejecución del programa.

(Formato)

TRON

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Al ejecutar la sentencia TRON, el BASIC se pone en modo “traza”, de manera que cada número de línea sale a pantalla antes de ejecutar la sentencia de dicha línea.
- Una vez ejecutada la sentencia TRON, el BASIC permanece en modo traza hasta que se ejecute TROFF o NEW.

(Ejemplo)

```
10 TRON
20 I=1
30 PRINT I
40 I=I*2
50 IF I>=1024 THEN TROFF:END
60 GOTO 30
70 END
```

Los números de línea de las líneas ejecutadas se presentan en pantalla entre corchetes "[]". Sin el PRINT I de la línea 30, las líneas de traza continuarían más hacia la derecha.

```
[20][30] 1
[40][50][60][30] 2
[40][50][60][30] 4
[40][50][60][30] 8
[40][50][60][30] 16
[40][50][60][30] 32
[40][50][60][30] 64
[40][50][60][30] 128
[40][50][60][30] 256
[40][50][60][30] 512
[40][50]
Ready
```


3.2.32. TROFF (desactivar trazas)

(Función)

La sentencia TROFF desactiva las trazas de la ejecución de un programa.

(Formato)

TROFF

(Versiones)

V1.0

V2.0

V3.0

(Ejemplo)

Ver Sección 3.2.31.

3.2.33. CHAIN (encadenar)

(Función)

La sentencia CHAIN llama y ejecuta un programa de un fichero especificado. El programa residente en memoria puede transferir todas las variables al programa llamado.

(Formato)

CHAIN[MERGE]“descriptor de fichero”
[, [expresión de número de línea][, ALL][, DELETE rango]]

(Versiones)

V2.0

V3.0

(Descripción)

- El descriptor de fichero especifica el programa llamado.
- El número de línea especifica el número de línea en el cual comienza el programa llamado. Cuando se omite este operando, el programa se ejecuta desde la primera línea. Se puede especificar tanto un número de línea como una expresión. Si se especifica una expresión, esta se evalúa, y el resultado es el número de la línea de arranque. El número de línea expresado por la expresión de número de línea no cambia al ejecutar un comando RENUM.
- Especificando ALL se transfieren todas las variables y matrices en curso. Cuando se omite este operando, solamente se transfieren las variables y matrices que se hayan especificado con la sentencia COMMON. Por lo tanto, se usa la sentencia COMMON cuando se desea transferir solamente las variables y matrices necesarias.

- Especificando MERGE se consigue que el programa en memoria y el llamado se junten. A continuación comienza la ejecución desde la línea especificada. Cuando se especifica MERGE, el programa llamado puede estar tanto en ASCII como en binario. Sin embargo, si el fichero está en binario, su primer número de línea debe ser mayor que el mayor número de línea del programa en memoria. Si no, se produce un error "Bad File Mode" (modo de fichero incorrecto). Especificando MERGE los ficheros utilizados por el programa en curso permanecen abiertos después de la ejecución del CHAIN.
- La opción DELETE solamente es válida cuando se especifica MERGE y ocasiona que las líneas dentro de un rango especificado se borren antes de juntarse. El rango (o la gama) se especifica del mismo modo que en la sentencia DELETE. Los números de línea especificados por DELETE se pueden cambiar con la sentencia RENUM.
- Cuando se omite MERGE, todas las definiciones de tipos de variables y de funciones de usuario especificadas en el programa precedente se invalidan. Por lo tanto, si es necesario, se deben reejecutar en el programa encadenado las sentencias DEFINT, DEFSNG, DEFDBL, DEFSTR y DEFFN.

(Ejemplo)

Programa principal

```
10 A=123
20 B=456
30 C$="F-BASIC"
40 CHAIN "0:SUB",,ALL
50 END
```

El programa principal asigna valores a las variables A, B, y C\$, a continuación ejecuta un subprograma usando estos valores como argumentos.

Subprograma

```
10 PRINT "A=";A
20 PRINT "B=";B
30 PRINT "C$=";C$
40 END
```

Ready

RUN

A=123

B=456

C\$=F-BASIC

Ready

3.2.34. COMMON(común)

(Función)

La sentencia COMMON transfiere variables a un programa encadenado con la sentencia CHAIN.

(Formato)

COMMON nombre de variable[,nombre de variable]...

(Versiones)

V2.0

V3.0

(Descripción)

- La sentencia COMMON especifica las variables a transferir al programa encadenado con una posterior sentencia CHAIN. Ya que la sentencia COMMON es una sentencia no ejecutable, puede aparecer en cualquier lugar del programa. No obstante, a efectos de documentación, es una buena práctica colocar las sentencias COMMON cerca del principio del programa.
- No debe especificarse el mismo nombre de variable en dos o más sentencias COMMON. Cuando se especifica una variable matriz, han de agregarse paréntesis () después del nombre de la variable.
- Al transferir todas las variables al programa encadenado con la sentencia CHAIN, es preferible especificar ALL en la sentencia CHAIN en vez de escribir una sentencia COMMON.
- La sentencia COMMON solamente es válida cuando no se especifica la opción ALL en la sentencia CHAIN asociada. Si se especifica un nombre de matriz no definido, se produce un error de "Illegal Function Call" (Llamada ilegal a función). El resultado de esto es el mismo que si se ejecutara la sentencia CLEAR y se inicializara la matriz.

(Ejemplo)

Programa principal

```
10 A=123
20 B=456
30 C$="F-BASIC"
40 COMMON A,C$
50 CHAIN "0:SUB"
60 END
```

Subprograma

```
10 PRINT "A=";A
20 PRINT "B=";B
30 PRINT "C$=";C$
40 END
```

Ready

RUN

A= 123

B=0

C\$=F-BASIC

Ready

3.2.35. ERASE (borrar)

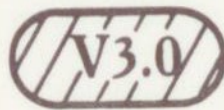
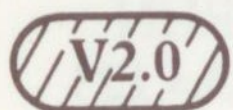
(Función)

La sentencia ERASE borra las variables matriz del programa.

(Formato)

ERASE nombre de variable matriz[,nombre de variable matriz]...

(Versiones)



(Descripción)

- Después de borrar una matriz, se puede declarar con la sentencia DIM una nueva matriz con el mismo nombre. Si no se ha borrado una matriz con la sentencia ERASE y se declara de nuevo con la sentencia DIM, se produce un error de definición duplicada ("Duplicate Definition").
- Cuando se borra una matriz, se libera el área de memoria que ocupaba, y puede ser utilizada para otro propósito.

(Ejemplo)

```
10 DIM A(10)
20 DIM B(10,20)
30 ERASE A
40 DIM A(30)
50 DIM B(10)
```

Como en la línea 30 la matriz A se elimina usando la sentencia ERASE, puede volver a declararse en la línea 40. Sin embargo, B no se borra, por lo que se produce un error en la línea 50.

Ready
RUN

Duplicate Definition In 50
Ready

3.3. Sentencias de entrada/salida

3.3.1. INPUT (entrada)

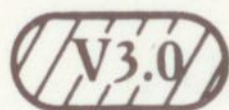
(Función)

La sentencia INPUT lee datos introducidos desde el teclado.

(Formato)

INPUT["mensaje de atención" {;|,}] nombre de variable
[, nombre de variable]...

(Versiones)



(Descripción)

- La sentencia INPUT hace que el programa se detenga y solicite al usuario, mostrándole el punto de interrogación (?), que introduzca datos desde el teclado.
- Opcionalmente se puede obtener un mensaje antes del punto de interrogación, y si el mensaje está seguido por una coma (,), aparecerá en pantalla sin punto de interrogación (?).
- Se debe introducir un dato por cada nombre de variable y su tipo debe concordar con el de la variable. Si los dos tipos son diferentes, o el número de datos introducidos es mayor o menor que el número de variables, aparece un mensaje "Redo From Start" (repetir desde el principio) y el programa espera la entrada de nuevo.
- Los datos se deben introducir separados por comas (,) o por el signo de dos puntos (:).

- Si se desea incluir una coma (,) o el signo de dos puntos (:) en una cadena de caracteres, o espacios delante o detrás de la misma, la cadena se ha de especificar entre comillas ("). En los demás casos, no es necesario utilizar comillas.
- Introduciendo **CTRL C** , **CTRL X** o pulsando la tecla BREAK (tecla STOP en las versiones **V1.0** y **V2.0**), en vez de introducir datos, provoca que el BASIC detenga la ejecución del programa y vuelva al nivel de comandos. Si a continuación se introduce un comando CONT, la ejecución del programa se reanuda desde el principio de la sentencia INPUT.

(Ejemplo)

```

10 S=0
20 INPUT "NUMERO DE DATOS?";I
30 FOR J=1 TO I
40 INPUT "DATO?";DA
50 S=S+DA
60 NEXT J
70 H=S/I
80 PRINT "MEDIA=";H
90 END

```

Ready

RUN

NUMERO DE DATOS? 3

DATO? 1

DATO? 2

DATO? 3

MEDIA = 2

3.3.2. LINE INPUT (entrada de línea)

(Función)

La sentencia LINE INPUT lee una línea completa de caracteres (255 caracteres o menos) y la almacena dentro de una variable cadena sin necesidad de utilizar caracteres delimitadores.

(Formato)

LINE INPUT["mensaje de atención" ; |, }]variable cadena

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Si se especifica un mensaje de atención, éste aparece preguntando por los datos.
- No aparece un punto de interrogación (?) para solicitar entrada de datos.
- Todos los caracteres tecleados se introducen en la variable cadena especificada.
- Introduciendo **CTRL C** , **CTRL X** o pulsando la tecla BREAK (tecla STOP en las versiones **V1.0** y **V2.0**), en vez de introducir datos, el BASIC suspende la ejecución de la sentencia LINE INPUT y vuelve al nivel de comandos. Si a continuación se introduce un comando CONT, la ejecución se reanuda desde el principio de la sentencia LINE INPUT.

(Ejemplo)

```
10 LINE INPUT "DATO ";A$  
20 IF A$="FIN" OR A$ "fin" THEN 50  
30 PRINT A$  
40 GOTO 10  
50 END
```

La sentencia LINE INPUT permite que se introduzcan como datos los delimitadores que se usan en la sentencia INPUT.

Ready

RUN

DATO 12.36

12.36

DATO "ABC",DEF,"GHI"

"ABC",DEF,"GHI"

DATO FIN

3.3.3. PRINT (escribir)

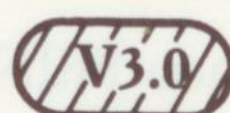
(Función)

La sentencia PRINT saca en pantalla el resultado de expresiones.

(Formato)

`PRINT[expresión[{/;}[expresión]]...]`

(Versiones)



(Descripción)

- La expresión debe ser una expresión numérica o de cadena. El valor resultante de la expresión se visualiza en la pantalla.
- Las expresiones deben separarse con comas (,), punto y coma (;), o espacios. Un espacio tiene el mismo efecto que un punto y coma. La excepción a esta regla es que puede omitirse el separador entre una variable y una constante de cadena, o también entre dos constantes de cadena. En estos casos se produce el mismo espaciado que cuando se usa punto y coma.
- El formato de salida viene determinado por la forma en que se separan las expresiones. Si dos expresiones están separadas por un punto y coma o un espacio, la salida de la segunda expresión aparece unida al resultado de la primera expresión sin espacio. Sin embargo, un valor numérico tiene un espacio delante y otro detrás cuando aparece en pantalla. Cuando la expresión numérica es negativa, el espacio de delante queda ocupado por un signo menos (-). Si dos expresiones están separadas por comas, la línea se divide en campos de catorce caracteres, y la salida de la segunda expresión comienza a partir de la primera posición del siguiente campo. Si el valor precedente ocupa dos o más campos, el siguiente valor comienza en el primer campo que sigue al campo en el cual termina el valor precedente. Un valor numérico tiene un espacio en blanco delante si es positivo, y un signo menos (-) si es negativo.

- Si la última expresión termina en coma o punto y coma, la siguiente sentencia PRINT escribe los datos en la misma línea, con el espaciado de acuerdo con las reglas anteriores. Si la última expresión no termina ni en coma ni en punto y coma, se produce un salto de línea y un retorno de carro.
- Si una línea de datos de salida es más larga que el ancho de la pantalla, continúa escribiéndose en la siguiente línea. Si una cadena o un valor numérico no cabe entre la posición del cursor y el final de la línea, se escribe en la siguiente línea (se produce un salto de línea).
- Una sentencia PRINT sin expresiones produce un salto de línea.
- Se puede introducir un punto de interrogación (?) en vez de la palabra clave PRINT para especificar esta función.

(Ejemplos)

```

10 A$="PRINT"
20 B=1234:C$="FBASIC":D$="BASIC"
30 PRINT A$
40 PRINT
50 PRINT C$,D$
60 PRINT B,C$;D$
70 PRINT A$;"ED"
80 END

```

Ready

RUN

PRINT

F-BASIC

1234

PRINTED

BASIC

F-BASICBASIC

3.3.4. LPRINT (imprimir)

(Función)

La sentencia LPRINT imprime en la impresora el resultado de evaluar cero o más expresiones.

(Formato)

LPRINT[expresión[{, | ; } [expresión]]...]

(Versiones)

V2.0

V3.0

(Descripción)

- La sentencia LPRINT hace exactamente lo mismo que la sentencia PRINT, excepto que los datos salen a la impresora en vez de a la pantalla.

(Ejemplo)

```
10 FOR I=&H20 TO &H3F
20 LPRINT CHR$(I);
30 NEXT
```

!"#\$%&'()*+,-./0123456789:;<=>?

3.3.5. PRINT USING (escribir con formato)

(Función)

La sentencia PRINT USING visualiza en pantalla cadenas de caracteres y números con el formato especificado.

(Formato)

PRINT USING "cadena de formato"; lista de salida

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

La lista de salida debe ser una lista de expresiones cadena o numérica separadas por punto y coma (;) o comas (,). Se pueden especificar los siguientes caracteres de control de formato en la cadena de formato. Estos caracteres de control no se visualizan en la pantalla.

Caracteres de control de formato de cadena

(1) *!* (punto de admiración)

Se especifica un punto de admiración para sacar solamente el primer carácter de una cadena de caracteres dada.

(2) *\n espacios *

Especificado *n* espacios ($n > 0$) entre dos barras invertidas, los primeros $n + 2$ caracteres de la cadena salen a la pantalla, y el resto se ignoran. Si la cadena es más corta que la longitud especificada ($n+2$), los caracteres se ajustan a la izquierda y el resto se rellena con espacios.

- (3) & (*ampersand*) ...válido en **V2.0** y **V3.0**.

Se especifica & para definir un área de cadena de longitud variable. La correspondiente cadena se escribe sin cambios.

Caracteres de control de formato numérico

- (1) # (*signo de número*)

El número de dígitos usados para visualizar en pantalla el número dado, se especifica repitiendo el signo de número (#) tantas veces como se desee. El número se ajustará a la derecha, y el resto se rellena con espacios. Un número negativo aparece precedido por el signo menos, que se cuenta como un dígito más y que necesita un signo de número (#).

- (2) . (*punto decimal*)

Se puede especificar un punto decimal en una posición arbitraria en un campo numérico. Si el punto decimal va seguido de signos de número (#), se presentan en pantalla tantos dígitos como veces se haya repetido el signo de número (#) después del punto decimal.

- (3) + (*más*)

Se especifica un signo más antes o después de la cadena de caracteres de control de formato numérico para añadir un signo (+ o -) delante o detrás del número.

- (4) - (*menos*)

Se especifica un signo menos al final de la cadena de caracteres de control de formato numérico para añadir un signo menos al final de un número negativo.

(5) ** (*dos asteriscos*)

se especifican dos asteriscos al principio de la cadena de caracteres de control de formato numérico para reemplazar con asteriscos todos los espacios a la izquierda del área numérica. Los dos asteriscos se cuentan como dígitos en el área numérica.

(6) \$\$ (*dos signos de dólar*)

Se especifican dos signos de dólar al principio de la cadena de caracteres de control de formato numérico para insertar un signo de dólar en el espacio inmediatamente precedente al número que se va a presentar. Los dos signos de dólar se cuentan como dos dígitos en el área numérica, y uno de ellos se usa para visualizar el signo de dólar.

(7) **\$ (*dos asteriscos y un signo de dólar*)

Se especifican dos asteriscos y un signo de dólar para llevar a cabo las funciones (5) y (6) a la vez. Estos tres símbolos se cuentan como tres dígitos en el área numérica, y uno de ellos se usa para visualizar el signo de dólar.

(8) , (*coma*)

Se especifica una coma a la izquierda del punto decimal para delimitar la parte entera con comas cada tres dígitos. Cada vez que se presenta una coma, se reserva una posición de un dígito.

(9) ^^^^ (*cuatro flechas*)

Si se especifican cuatro signos de flecha después de los signos # indicando el número de dígitos, el resultado se representa en forma exponencial. Los cuatro símbolos indican los cuatro dígitos usados para representar E+nn o D+nn, donde nn varía según el número de dígitos significativos indicado por los signos de número (#).

(10) *_(signo de subrayado) válido en (V1.0) y (V2.0)*

Se especifica el signo de subrayado para conseguir que el siguiente carácter salga tal cual (sin ser considerado carácter de control de formato). Para obtener el propio signo de subrayado, se especifican dos signos de subrayado.

Visualización de cadenas

Dentro de la cadena de formato se pueden escribir caracteres diferentes de los de control de formato. Estos caracteres aparecen en la pantalla antes y después de aquellos caracteres cuyo formato se haya cambiado.

Nota:

Si un número no puede ser visualizado en pantalla con el número especificado de dígitos, aparece un signo (%) al principio del número. Del mismo modo, si se redondea un resultado en más dígitos de los especificados, aparece un signo de tanto por ciento delante del número redondeado.

(Ejemplos)

```
10 A=12345:B=62.35
20 C=-160
30 D=30.251E+12
40 PRINT USING "#####.##";A;A
50 PRINT USING "+#####.##";B;C
60 PRINT USING "**##.###$$#####";B;C
70 PRINT USING "**$###;D ##$####,";B;A
80 PRINT USING "##### ^^^";D
```

Ready
RUN

12345	12345.0
+62	160-
**62.350	-\$160
*****\$62	*\$12,345
3251E+09	

Ready

```
10 A=65537!
20 PRINT USING "#####-!";A
```

Ready
RUN
65537!

Debido al signo de subrayado en la sentencia de PRINT USING en la línea 20, el punto de admiración no se considera como carácter de control de formato.

Ready

```
10 A$="BASIC":B$="FBASIC":C$="BASIC"
20 PRINT USING " \ \ \ \ !";A$;B$;C$
30 END
```

Ready
RUN
BASIC F-BASIC B

Ready

3.3.6. LPRINT USING (imprimir con formato)

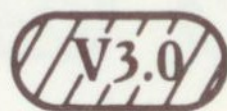
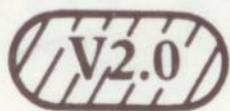
(Función)

La sentencia LPRINT USING saca a la impresora números o caracteres con el formato especificado.

(Formato)

LPRINT USING cadena de formato;lista de salida

(Versiones)



(Descripción)

El uso de LPRINT USING es idéntico al de la PRINT USING, excepto en que la salida se dirige a la impresora en vez de a la pantalla.

(Ejemplos)

```
10 A=123
20 B=567
30 C$="FUJITSU"
40 LPRINT USING "##### ###.#### ";A;B
50 LPRINT USING "&";C$
60 END
```

```
123 567.0000
FUJITSU
```


3.3.7. OPEN (abrir)

(Función)

La sentencia OPEN realiza el proceso de apertura de ficheros.

(Formato)

OPEN "modo", [#] número de fichero, "descriptor de fichero"

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia asigna un número de fichero y un buffer de entrada/salida al fichero indicado por el descriptor de fichero; a partir de este momento, se pueden especificar operaciones de entrada/salida con dicho número de fichero.
- Ha de ejecutarse siempre una sentencia OPEN antes de una operación de entrada/salida de datos en un fichero con las sentencias INPUT#, PRINT#, LINE INPUT#, GET, o PUT, o antes de usar cualquier función que requiera número de fichero.
- Debe especificarse el modo de entrada/salida del fichero con uno de los siguientes caracteres: I, O, A o R.
 - I: El fichero especificado se abre para entrada. El nombre del fichero especificado debe estar presente en el dispositivo especificado.
 - O: El fichero especificado se abre para salida. El nombre del fichero especificado debe estar presente en el dispositivo especificado.
 - A: Se abre un fichero secuencial para salidas adicionales. El fichero especificado debe estar presente en el disco. Esta especificación sólo es válida para ficheros en disco.

R: El fichero de acceso directo especificado se abre para procesos de entrada/salida. Esta especificación sólo es efectiva para ficheros en disco. Si el nombre del fichero especificado no se encuentra en el disco, se registra como un nombre de fichero nuevo.

- El número de fichero debe estar comprendido entre 1 y 16.
- Un fichero en disco se puede abrir con más de un número de fichero. No obstante, su modo de entrada/salida debe ser I o R.
- Si el dispositivo es un cassette de burbuja o una cassette de audio, sólo se puede abrir un fichero a la vez en la misma unidad.
- Se pueden especificar opciones en el descriptor de fichero para la interfaz RS-232C o la impresora. Para la impresora, puede especificarse S o W para indicar el número de dígitos a imprimir. S establece el número de dígitos en 80, y W en 136. Una vez especificada la opción permanece válida hasta que se especifique la otra opción. Esta opción se pone a S al arrancar el BASIC. Para las opciones del RS-232C, ver la Subsección 3.8.3.

(Ejemplos)

```
10 OPEN "O",#1,"0:DATA"  
20 READ A$  
30 IF A$="E" THEN 90  
40 READ B  
50 PRINT #1,A$  
60 PRINT #1,B  
70 GOTO 20  
80 DATA IC,200,PERSONAL  
  COMPUTER,300  
85 DATA MICROCOMPUTER,  
  6809,E  
90 CLOSE #1  
100 STOP  
110 PRINT "RESUMEN DE VENTAS"  
120 OPEN "I",#2,"0:DATA"  
130 FOR I=1 TO 3  
140 INPUT #2,A$  
150 INPUT #2,B  
160 PRINT A$,B  
170 NEXT I  
180 END
```

Las líneas 10 a 90 crean un fichero secuencial en el disco flexible. La ejecución del programa se detiene en la línea 100. Puede usarse el comando CONT para restablecer la ejecución, con lo que las líneas 110 a 170 leen datos del fichero secuencial y los visualizan en pantalla.

3.3.8. CLOSE (cerrar)

(Función)

La sentencia CLOSE realiza el proceso de cierre de ficheros.

(Formato)

CLOSE[[#]número de fichero[, [#]número de fichero]...]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia revoca la asignación de un número de fichero a un descriptor de fichero para permitir a otros ficheros el uso del buffer de entrada/salida.
- Una sentencia CLOSE ha de ser ejecutada siempre al final de los procesos de entrada/salida que usan una sentencia INPUT# o PRINT#.
- La ejecución de la sentencia CLOSE sin ningún número de fichero cierra todos los ficheros abiertos.
- La ejecución de la sentencia END también cierra automáticamente todos los ficheros abiertos. Por el contrario, la sentencia STOP no cierra los ficheros.

(Ejemplos)

Ver la Sección 3.3.7.

3.3.9. INPUT# (entrada por fichero)

(Función)

La sentencia INPUT# lee datos de un fichero y los asigna a una variable.

(Formato)

INPUT#número de fichero,nombre de variable[,nombre de variable]...

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Ha de especificarse el número de fichero usado cuando se abrió el fichero en modo entrada (modo "I") con una sentencia OPEN.
- El nombre de variable especifica el área en la cual se guardan los datos leídos. El tipo del dato leído y el de la variable han de coincidir. Cuando se lee el dato y se carga dentro de una variable de cadena, las comas, dos puntos, retornos de carro y saltos de línea son tratados como separadores. Los caracteres nulos antes y después de los datos se ignoran. Para incluir estos caracteres como datos significativos, han de ser especificados entre comillas ("). Obsérvese que los datos colocados entre comillas no pueden contener comillas. Cuando un dato se lee y se carga dentro de una variable numérica, las comas, dos puntos, retornos de carro y saltos de línea son tratados como separadores. Los caracteres nulos antes y después de los datos se ignoran.

(Ejemplos)

Ver la Sección 3.3.7.

3.3.10. PRINT# (escribir en fichero)

(Función)

La sentencia PRINT# saca el resultado evaluado de expresiones a un fichero especificado.

(Formato)

PRINT#número de fichero[,lista de salida]

o

PRINT#número de fichero,USING cadena de formato;lista de salida

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Ha de especificarse el número de fichero usado cuando se abrió el fichero en el modo de salida (modo "O") o el modo de salida adicional (modo "A") con una sentencia OPEN.
- La lista de salida es una lista de expresiones numéricas o de cadena cuyos valores se van a escribir en el fichero. Dos expresiones o más deben separarse por punto y coma (;) o por comas (,).
- Para sacar datos de cadena a un disquete, se especifica una coma (,) al final. Sin esta coma, si la cadena va seguida de otra cadena, las dos cadenas se concatenan en una, haciendo que una sentencia INPUT las lea como una sola cadena de datos. Cuando se sacan datos de cadena a un cassette de burbuja o cassette audio, no hace falta poner la coma ya que sale un código CR (retorno de carro) después de cada dato.
- Para escribir comillas ("), se usa CHR\$(34).

(Ejemplo)

```
10 OPEN "O",#1,"LPT0:"  
20 INPUT "EL NOMBRE ES";A$  
30 PRINT #1,A$;  
40 INPUT "EL TELEFONO ES";B$  
50 PRINT #1,"          ";B$  
60 CLOSE #1
```

Este programa saca a la impresora datos que fueron introducidos mediante el teclado.

3.3.11. LINE INPUT# (entrada de línea de fichero)

(Función)

La sentencia LINE INPUT# lee una línea entera de un fichero especificado y la asigna a una variable de cadena sin caracteres separadores.

(Formato)

LINE INPUT# número de fichero, variable de cadena

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se especifica el número de fichero usado cuando se abrió el fichero para entrada (modo "I") con una sentencia OPEN.
- La variable de cadena debe ser un nombre de variable de cadena al cual se asigna una línea entera.
- Esta sentencia lee todos los caracteres (máximo: 255) hasta el código CR y los guarda en la variable de cadena especificada, ignorando los separadores. Cualquier otro código de control detectado antes de CR no será leído.

(Ejemplos)

```
10 OPEN "O,#1,0:DATA"
20 READ A$
30 IF A$="E" THEN 80
40 READ B
50 PRINT#1,A$,B
60 GOTO 20
70 DATA JACK,100,MARY,101,JIM,102,E
80 CLOSE#1
90 STOP
100 PRINT"NOMBRE          NO."
110 PRINT
120 OPEN "I",#2,"0:DATA"
130 LINE INPUT#2,A$
140 PRINT A$
150 IF EOF (2) THEN 170
160 GOTO 130
170 CLOSE#2
180 END
```

RUN

Break in 90

Ready

CONT

NOMBRE	NO.
--------	-----

JACK	100
------	-----

MARY	101
------	-----

JIM	102
-----	-----

Ready

3.3.12. FIELD (campo)

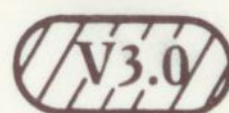
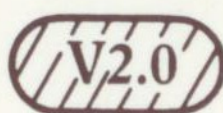
(Función)

La sentencia FIELD asigna áreas de variable en un buffer para un fichero directo.

(Formato)

FIELD[#]número de fichero, ancho de campo AS
nombre de variable cadena[, ancho de campo
AS nombre de variable cadena]...

(Versiones)



(Descripción)

- Cuando se realiza una operación de entrada/salida en un fichero directo utilizando las sentencias GET o PUT, primero hay que ejecutar esta sentencia FIELD para dividir el buffer del fichero directo en campos en los cuales se almacenen los datos de cada variable. Un buffer ocupa 256 bytes, de modo que el ancho total de los campos de todas las variables cadena no debe exceder de 256.
- El número de fichero es el especificado cuando se abrió el fichero en el modo de entrada/salida directo (modo "R").
- Se especifica número de fichero 0 para definir un buffer directo del sistema que se use con DSKI\$ o DSKO\$.
- La longitud del campo, indica el número de caracteres asignados a una variable cadena.
- Una variable cadena que aparece en una sentencia FIELD debe definirse con una sentencia LSET o RSET, y no con una sentencia INPUT ni de asignación (LET).

- Se pueden ejecutar dos sentencias FIELD o más para el mismo número de fichero. Cada sentencia FIELD redefine el buffer desde el principio. Por lo tanto, se puede hacer referencia al mismo campo con distintos nombres de variable.

(Ejemplo)

```
10 OPEN "R",#1,"DATA"  
20 FIELD #1,20 AS A$,30 AS B$,40 AS C$  
30 FIELD #1,90 AS DUMMY$,4 AS D$,4 AS E$  
40 GET #1
```

El contenido de DUMMY\$ es igual a la suma de A\$, B\$ y C\$.

3.3.13. GET (obtener)

(Función)

La sentencia GET lee un registro especificado de un fichero directo, y lo carga en un buffer.

(Formato)

GET[#]número de fichero[,número de registro]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se especifica el número de fichero usado cuando se abrió el fichero en el modo de entrada/salida directo (modo "R").
- El número de registro indica la posición del registro que se va a leer en el fichero directo, y su valor puede variar entre 1 y el número de registro máximo. El tamaño del registro que se va a leer es de 256 bytes. Si se omite el número del registro, se lee el registro que sigue al leído o escrito por la última sentencia GET o PUT ejecutada para ese fichero.

(Ejemplo)

```
10 OPEN "R",#1,"DATA1"  
20 FIELD #1,20 AS NAM$,20 AS  
   TEL$  
30 INPUT "NUMERO DE  
   REGISTRO";A  
40 IF A>100 THEN 110  
50 GET #1,A  
60 A$=NAM$  
70 B$=TEL$  
80 PRINT A$,B$  
90 PRINT  
100 GO TO 30  
110 CLOSE #1  
120 END
```

Este programa lee datos de un fichero directo de disquete que se llama DATA1 y los saca a la pantalla.

3.3.14. PUT (poner)

(Función)

La sentencia PUT escribe el contenido de un buffer en un fichero directo.

(Formato)

PUT[#]número de fichero[,número de registro]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Especificar el número de fichero usado cuando se abrió el fichero en el modo de entrada/salida directo (modo "R") por una sentencia OPEN.
- El número de registro identifica un registro en un fichero directo en el cual se escriben datos, y puede variar entre 1 y nnn.
- Si se omite el número del registro, los datos son sacados al registro que sigue al leído o escrito mediante la última sentencia GET o PUT ejecutada para ese fichero.

(Ejemplo)

```
10 OPEN "R",#1,"DATA1"
20 FIELD#1,20 AS NAM$,20 AS
   TEL$
30 INPUT "NOMBRE ";A$
40 IF A$="FIN" THEN 100
50 INPUT "TEL ";B$
60 LSET NAM$=A$
70 LSET TEL$=B$
80 PUT #1
90 GOTO 30
100 CLOSE #1
110 END
```

Este programa escribe los datos introducidos por teclado en un fichero directo llamado DATA1.

3.3.15. DSKO\$ (salida a disco \$)

(Función)

La sentencia DSKO\$ escribe el contenido de un buffer directo del sistema en un sector especificado.

(Formato)

DSKO\$ número de unidad, número de pista,número de sector

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La variable cadena que se va a asignar al buffer directo del sistema debe definirse por una sentencia FIELD antes de ejecutar esta sentencia.
- El número de unidad debe estar comprendido entre 0 y 3.
- El número de pista debe estar entre 0 y 39.
- Si se utiliza un minidisquete flexible, el número de sector debe estar entre 1 y 32. Los números 17 a 32 corresponden a los sectores 1 al 16 en la segunda cara del disco. Si se usa un disco estándar, el número de sector debe estar entre 1 y 52. Los números 27 a 52 corresponden a los sectores 1 a 26 en la segunda cara del disco.
- El buffer directo del sistema es un área de 256 bytes en la memoria.

(Ejemplo)

```
10 FIELD #0,128 AS A$,128 AS B$
20 DUMMY$=DSKI$(0,10,1)
30 DSKO$ 0,39,1
```

Este programa lee el contenido del sector 1, pista 10 de la unidad 0, y lo escribe en el sector 1 de la pista 39 de la misma unidad, es decir la 0.

3.4. Control de pantalla y funciones gráficas

3.4.1. WIDTH (ancho)

(Función)

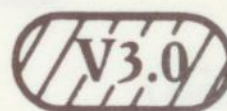
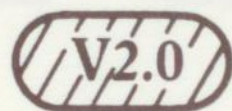
La sentencia WIDTH especifica el número de líneas y columnas en la pantalla.

(Formato)

WIDTH [número de caracteres por línea][, número de líneas por pantalla]

(Abreviado W.)

(Versiones)



(Descripción)

- Se pueden elegir entre 40 u 80 caracteres por línea.
- Se pueden elegir entre 20 ó 25 líneas por pantalla.
- Cuando se ejecuta una sentencia WIDTH, la pantalla se borra.
- Cuando se pone en marcha el BASIC, la pantalla se activa inicialmente con 40 columnas por 20 líneas.
- La ejecución de WIDTH, 20 implica la ejecución de CONSOLE 0, 20, 0, 0 (ver la Sección 3.4.2).

(Ejemplo)

WIDTH 80,25

La pantalla se establece con 80 caracteres por 25 líneas.

3.4.2. CONSOLE (consola)

(Función)

La sentencia CONSOLE especifica el tamaño de la ventana con SCROLL dentro de la pantalla.

(Formato)

CONSOLE[línea de comienzo con scroll]
[, [número de líneas con scroll]
[, [indicador de tecla de función]
[, [indicador de color]]]

(Abreviado CONS.)

(Versiones)



(Descripción)

- Se puede seleccionar el tamaño de la ventana con scroll y así establecer el modo de página especificando los dos siguientes operandos:

Línea de comienzo con scroll:	0 a 24
Número de líneas con scroll:	0 a 25

Si la línea de comienzo de scroll se pone a 0 o en la 24 (19 en el caso de pantalla de 20 líneas), y el número de líneas con scroll se pone a 0, toda la pantalla queda en modo de página. Si la línea de comienzo de scroll se pone entre 1 y 23, y el número de líneas de scroll se pone a 0, la pantalla se divide en dos pantallas de una página. La mitad superior es la pantalla 1 en modo de página y la mitad inferior es la pantalla 2 en modo de página.

- Si el indicador de tecla de función se pone a 1, las dos líneas inferiores se usan para mostrar las definiciones de las teclas de función, de modo que quedan disponibles dos líneas menos que el tamaño de la ventana, para el scrolling. Si el indicador de teclas de función se pone a 0, no se visualizan en pantalla las definiciones de las teclas de función.

- El indicador de color de consola especifica si se visualizan los caracteres sólo en verde o en el color especificado. Si se pone a 0, se usa el color seleccionado con la sentencia COLOR. Si se pone a 1, solamente se usa el verde, lo cual acelera las operaciones de salida. No olvidar que si se usan gráficos en color y después se lista un programa con el indicador de consola puesto a 1 (modo monocromo), parte de la imagen de pantalla permanece en pantalla. Para evitar esto, pulsar la tecla CLEAR o ejecutar la sentencia CLS antes de hacer el listado.
- Al activar el BASIC, el tamaño de la ventana de scroll se establece inicialmente a CONSOLE 0, 20, 0, 0.

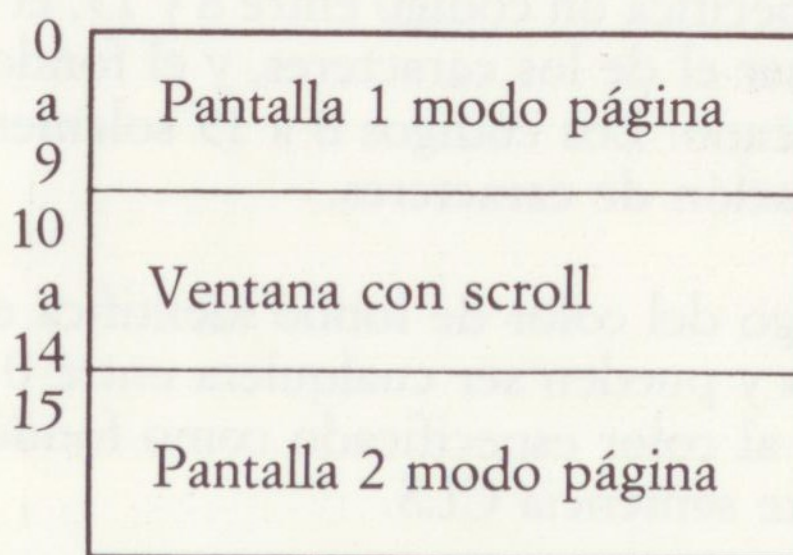
(Ejemplos)

CONSOLE 0,17,1,0

La ventana de scroll se sitúa entre las líneas 0 y 15, y la asignación de las teclas de función se visualiza en las dos últimas líneas.

CONSOLE 10,5,0,0

Las 10 líneas superiores de la pantalla 1 están en modo página, la ventana de scroll abarca las líneas 10 a 14, y las líneas desde la 15 hasta el final forman la pantalla 2 en modo página.



Pantalla

3.4.3. COLOR (color)

(Función)

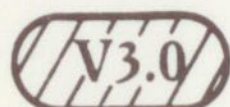
La sentencia COLOR especifica el color de los caracteres, de la visualización gráfica y del fondo de la pantalla.

(Formato)

COLOR[código de color][,código de color de fondo]

(Abreviado COL.)

(Versiones)



(Descripción)

- El código de color identifica el color en el cual se desea visualizar los caracteres y los gráficos.

0(8) ---- Negro

1(9) ---- Azul

2(10) ---- Rojo

3(11) ---- Morado

4(12) ---- Verde

5(13) ---- Azul celeste

6(14) ---- Amarillo

7(15) ---- Blanco

Si se especifica un código entre 8 y 15, el color que tenga el fondo pasa a ser el de los caracteres, y el fondo toma el color especificado. Los códigos 8 a 15 solamente son válidos para la visualización de caracteres.

- El código del color de fondo identifica el color del fondo de la pantalla y pueden ser cualquiera entre 0 y 7. La pantalla no cambia al color especificado como fondo hasta que se ejecuta la siguiente sentencia CLS.

- Al activar el BASIC, se inicializa la pantalla a COLOR 7,0.

(Ejemplos)

```
10 WIDTH 80,25:CLS
20 FOR I=1 TO 1920
30 C=CINT(RND(1)*6)+1
40 COLOR C
50 PRINT "■ ";
60 NEXT I
70 LOCATE 0,0
80 COLOR 7,0
90 END
```

El color de pantalla se determina con un número aleatorio.

(Formato 2)

COLOR[color en primer plano][color del fondo]

(Función)

Este formato especifica el color de lo visualizado y del fondo.

(Versiones)



(Descripción)

- El color del primer plano es el color en el cual se visualizan los caracteres. El color del fondo debe especificarse con el código de la paleta y no con el código de color. La relación entre el código de la paleta y el código de color se establece con la sentencia COLOR en formato 3. El color del primer plano se especifica normalmente con los códigos de paleta 0 al 7. Para video inverso, se añade 8 a estos códigos, y se especifica un número entre 8 y 15, intercambiando así los colores de los caracteres y del fondo.
- El color de fondo es el color con el que aparece el fondo. El color especificado no aparece hasta que se ejecuta la siguiente sentencia CLS después de la sentencia COLOR. El color del fondo, al igual que el color del primer plano, se especifica con los códigos de paleta 0 al 7.

- Al activar el BASIC, la pantalla se inicializa a COLOR 7,0.
- Si se omite el código de la paleta en una sentencia de gráficos, el color del primer plano utilizado es el de la última sentencia COLOR ejecutada.

(Ejemplos)

Ver formato 3.

(Formato 3)

COLOR=(código de paleta,código de color)

(Función)

Esta sentencia define la paleta de colores.

(Versiones)



(Descripción)

- Todas las especificaciones de color en las sentencias gráficas del F-BASIC se hacen con los códigos de paleta. Esta sentencia COLOR relaciona los códigos de color y de paleta. Al activar el BASIC, los códigos de paleta son idénticos a los códigos de color, tal como se indica a continuación.

Código de paleta

Código de color

0	-----	0 Negro
1	-----	1 Azul
2	-----	2 Rojo
3	-----	3 Morado
4	-----	4 Verde
5	-----	5 Azul celeste
6	-----	6 Amarillo
7	-----	7 Blanco

Por lo tanto, mientras no se ejecute una sentencia COLOR en formato 3, el código de paleta puede usarse igual que el código de color. La sentencia COLOR en formato 3 permite asignar arbitrariamente colores a la paleta de colores. Por ejemplo, se puede cambiar el azul por rojo y el rojo por azul, o hacer otras modificaciones libremente. Al ejecutar la sentencia COLOR, los colores de los caracteres y los gráficos presentados actualmente en pantalla con código de paleta, cambiarán instantáneamente a los nuevos colores especificados.

- El código de paleta 0 sólo puede asignarse al color negro.

(Ejemplos)

```
10 WIDTH 80,25
20 CLS
30 COLOR=(7,1)
40 COLOR 0,7
45 CLS
46 LINEA (220,50)-(420,150),PSET,1,BF
50 FOR I=1 TO 500
60 NEXT I
80 FOR I=2 TO 7
90 COLOR=(7,I)
100 LOCATE 0,0:PRINT "COLOR=(7,";I;")"
110 FOR J=1 TO 1000
120 NEXT J
130 NEXT I
140 FOR I=1 TO 7
150 COLOR=(I,I)
160 NEXT
170 END
```


3.4.4. SCREEN (pantalla)

(Función)

La sentencia SCREEN establece las VRAMs (RAM de video) activas y las VRAMs visualizadas.

(Formato)

SCREEN[código de VRAM activa]
[,código de VRAM visualizada]

(Versiones)



(Descripción)

- El código de VRAM activa especifica cuál de las VRAMs (B, R o G) puede aceptar datos para posterior visualización en pantalla. El código se especifica con un dígito entre 0 y 7. Los tres bits en la representación binaria del número especificado corresponden a B, R y G, y la VRAM activa se representa con un 1 binario. Se pueden escribir datos dentro de la VRAM cuyo bit sea 1, es decir, activa. No se pueden escribir datos en la VRAM cuyo bit sea 0, es decir, enmascarada (que protege su contenido actual).

Código de VRAM activa	Bit			Significado
	2	1	0	
	G R B			
0	0	0	0	Todas las VRAMs enmascaradas
1	0	0	1	G enmascarada, R enmascarada, B activa
2	0	1	0	G enmascarada, R activa, B enmascarada
3	0	1	1	G enmascarada, R activa, B activa
4	1	0	0	G activa, R enmascarada, B enmascarada
5	1	0	1	G activa, R enmascarada, B activa
6	1	1	0	G activa, R activa, B enmascarada
7	1	1	1	Todas las VRAMs activas

Cuando un código de VRAM activa se especifica para permitir y enmascarar la escritura de datos tal como se muestra arriba, hay que tener cuidado con los códigos de paleta en las sentencias posteriores. El código de paleta en curso determina qué VRAMs (B, R, o G) son capaces de aceptar datos válidos, y esas VRAMs deben ser también activas. Por lo tanto, haciendo un AND entre el código de VRAM activa y el código de paleta, el resultado no debe ser 0.

- El código de VRAM visualizado especifica cuál de las VRAMs (B, R, o G) envía sus datos a la pantalla. El código varía entre 0 y 7, y tiene el siguiente significado.

Código de
VRAM
visualizada

Significado

0	No se visualiza nada de la VRAM.
1	Solo se visualiza el azul.
2	Solo se visualiza el rojo.
3	Se visualiza la combinación azul y rojo.
4	Solo se visualiza el verde.
5	Se visualiza la combinación azul y verde.
6	Se visualiza la combinación rojo y verde.
7	Se visualiza toda la VRAM.

- Hay que tener cuidado al seleccionar el código de VRAM activa y el código de VRAM visualizada. Si no se tiene cuidado, uno se puede encontrar frente a una pantalla en blanco. Tal situación puede deberse a que no se ha escrito nada en la pantalla a causa de un error al seleccionar el código de VRAM activa y el código de paleta, o a que se estén visualizando los datos de un VRAM equivocada. Por ejemplo, la ejecución de COLOR 4:SCREEN 4,2 produce una pantalla completamente negra, en la que no aparece el Ready, aunque el BASIC sigue en el nivel de comandos. Cuando ocurre algo como esto, pulsando PF 9 el Código de VRAM activa y el de VRAM visualizada vuelven a los valores iniciales (Ya que PF 9 está definida como SCREEN 7,7.

(Ejemplos)

```
10 WIDTH 80,25
20 C.I.S
30 SCREEN 7,0
40 FOR I=1 TO 7
50 COLOR I,0
60 PRINT STRING$(80, "■")
70 NEXT I
80 FOR I=0 TO 7
90 SCREEN 0,I
100 FOR J=1 TO 2000
110 NEXT J
120 NEXT I
130 SCREEN 7,7
140 END
```


3.4.5. CLS (borrar pantalla)

(Función)

La sentencia CLS borra una porción especificada de la pantalla, y mueve el cursor a la esquina superior izquierda.

(Formato)

CLS[código de zona a borrar]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se puede especificar un código de zona a borrar entre 0 y 3 para obtener lo siguiente:

0: Se borra toda la pantalla.

1: Se borra la pantalla de scroll.

2: Se borra la pantalla 1 en modo de página.

3: Se borra la pantalla 2 en modo de página.

Si se omite el código de zona a borrar, se borra toda la pantalla.

(Ejemplos)

10 CONSOLE 10,0,0,0

20 CLS 2

.

.

.

80 CLS 3

CLS 2 borra la pantalla 1 en modo de página y mueve el cursor a la esquina superior izquierda de dicha pantalla. CLS 3 borra la pantalla 2 en modo de página, y entonces mueve el cursor a la esquina superior izquierda de dicha pantalla.

3.4.6. LOCATE (colocar)

(Función)

La sentencia LOCATE desplaza el cursor a una posición especificada de la pantalla.

(Formato)

LOCATE posición horizontal, posición vertical
[, indicador de visualización del cursor]

(Versiones)

V1.0

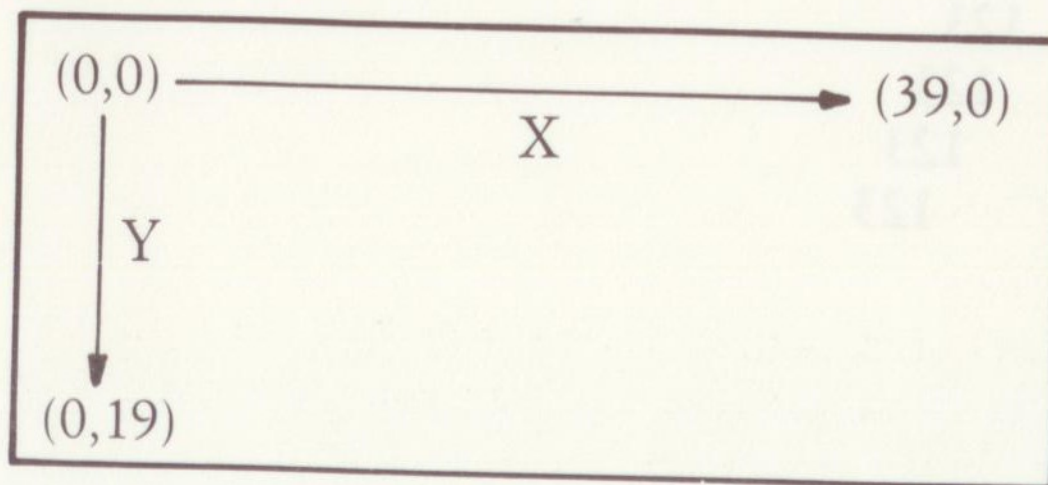
V2.0

V3.0

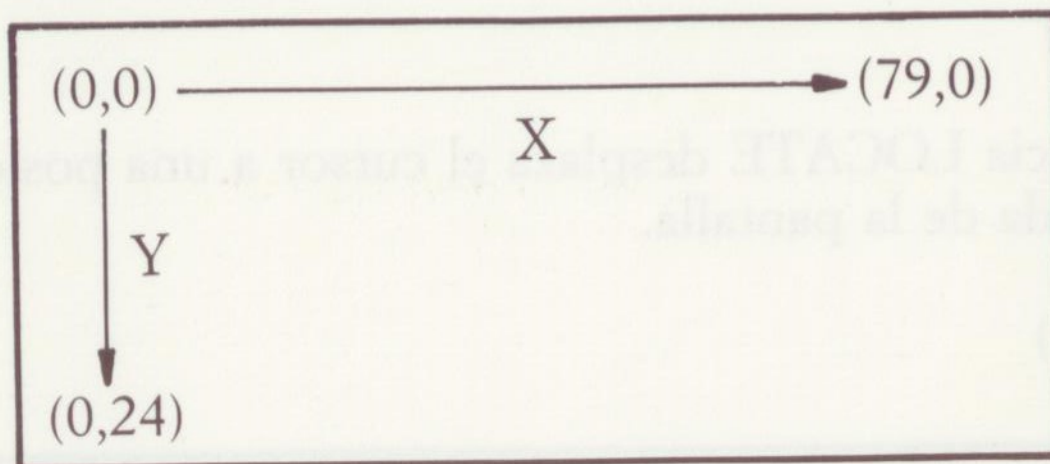
(Descripción)

- Se pueden especificar las coordenadas de la siguiente posición del cursor, indicando la posición horizontal y la vertical.
- Si el indicador de visualización del cursor se pone a 1, el cursor se visualiza en la posición especificada. Si se pone a 0, el cursor no se visualiza. Si se omite, el cursor se queda como se hizo en la anterior sentencia LOCATE ejecutada. (Inicialmente se pone a 1.)
- La gama de posiciones del cursor que se pueden especificar depende del modo de pantalla especificado con WIDTH.

(1) WIDTH 40,20



(2) WIDTH 80,25



Si la cadena de caracteres de las teclas de función se visualizan en las dos líneas inferiores de la pantalla, no se pueden especificar coordenadas del cursor en dichas líneas.

(Ejemplos)

```
10 CLS
20 FOR I=1 TO 10
30 LOCATE I,Y
40 PRINT "123"
50 Y=Y+1
60 NEXT I
70 END
```

```
123
 123
   123
    123
     123
      123
       123
        123
         123
          123
```



3.4.7. PSET (establecer punto)

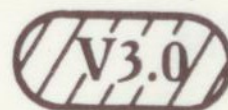
(Función)

La sentencia PSET establece un punto en una posición determinada de la pantalla.

(Formato)

PSET(posición horizontal,posición vertical
[, [código de paleta][,función]])

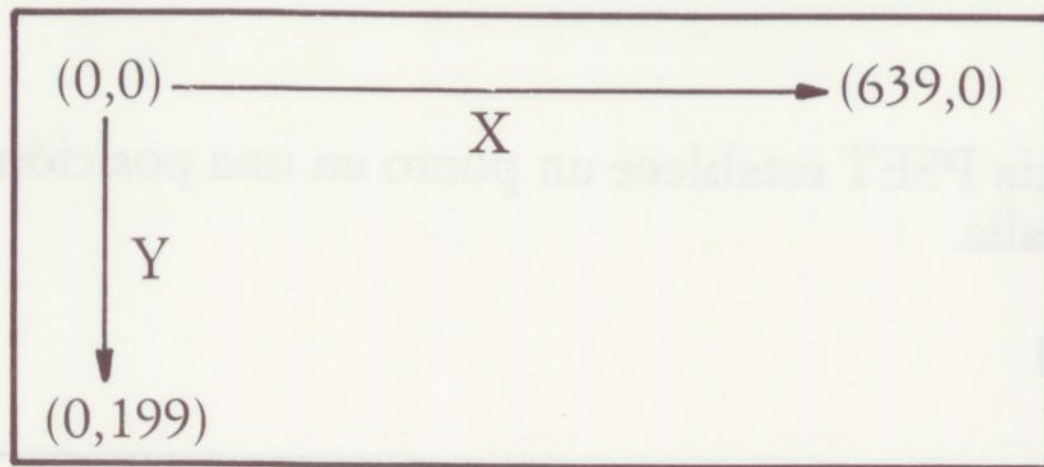
(Versiones)



(Descripción)

- Las posiciones horizontal y vertical se especifican como un par de coordenadas X,Y. Pueden ser números enteros o números reales. Con números reales, se utiliza el entero más próximo (considerando las fracciones de 0,5 ó mayores como 1, y las fracciones menores de 0,5 como 0). La posición horizontal puede estar entre 0 y 639, y la posición vertical entre 0 y 199.
- El código de paleta identifica el color del punto que se va a visualizar en la coordenada especificada. Si se omite, el punto se visualiza con el color del primer plano especificado en la última sentencia COLOR ejecutada.
- La función puede especificarse como AND, OR o XOR. Si se especifica una de estas funciones, se realiza la operación lógica entre el código de paleta del punto actualmente visualizado en la pantalla y el código de paleta especificado por el operando. El punto se visualiza con el color resultante.

- Coordenadas de los puntos en la pantalla.



(Ejemplos)

```
5  COLOR 2,0
10 CLS
20 PSET(100,50,2,XOR)
30 PSET(540,50,3,XOR)
40 PSET(540,150,4,XOR)
50 PSET(100,150,5,XOR)
60 END
```

Después de la ejecución de la sentencia PSET, se visualiza un punto rojo en las coordenadas (100,50), un punto morado en (540,50), un punto verde en (540,150) y un punto azul celeste en (100,150).

3.4.8. PRESET (borrar punto)

(Función)

La sentencia PRESET visualiza en pantalla un punto en la posición especificada y con el color del fondo.

(Formato)

PRESET(posición horizontal, posición vertical)

(Versiones)



(Ejemplos)

```
10 COLOR 7,0
20 CLS
30 PSET(320,100,2,OR)
40 FOR I=0 TO 200
50 NEXT I
60 PRESET(320,100)
70 FOR I=0 TO 200
80 NEXT I
90 GOTO 30
```

Se visualiza un punto rojo en (320,100) al combinar la sentencia PSET de la línea 30 y la sentencia PRESET de la línea 60.

3.4.9. LINE (línea)

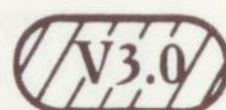
(Formato 1)

`LINE[@][(X1,Y1)]-(X2,Y2),cadena[, [código de paleta][,{B|BF}]]`

(Función)

La sentencia LINE en formato 1 visualiza una línea o un rectángulo formado por caracteres.

(Versiones)



(Descripción)

- Se puede dibujar una línea entre la posición (X₁,Y₁) y la posición (X₂,Y₂), formada por una repetición del primer carácter de una cadena dada. Los números X₁ y X₂ representan las coordenadas horizontales que pueden estar entre 0 y uno menos que el número de caracteres de la línea; Y₁ e Y₂ representan las coordenadas verticales, y pueden estar entre 0 y uno menos que el número de líneas de la pantalla.
- El código de paleta especifica el color con el cual se visualizan los caracteres.
- Especificando B se visualiza un rectángulo con (X₁,Y₁) y (X₂,Y₂) en las esquinas diagonalmente opuestas. Especificando BF se rellena el interior del rectángulo con los caracteres especificados.
- Si el código de paleta se omite, se usa el color de primer plano especificado en la última sentencia COLOR ejecutada.
- Si se omite (X₁,Y₁), se usa el (X₂,Y₂) especificado en la última sentencia LINE ejecutada.
- Si la cadena consiste en caracteres nulos, la línea se dibuja utilizando el código nulo (&H00).

(Ejemplos)

```

10 CLS
20 X=20:Y=5
30 LINE@ (X-15,Y)-(X,Y),"H"
40 LINE@ -(X+10,Y+5),"B",1,B
50 LINE@ (31,8)-(51,20),"C",2,BF
60 LINE@ (0,8)-(28,18),"N",3
70 LOCATE 0,21
80 END

```

	HHHHHHHHHHHHHHHHHHHHHHHH	BBBBBBBBBBBBBB	
		B	B
		B	B
NNN		B	BCCCCCCCCCCCCCCCCCCCCCCCC
NNNN		B	BCCCCCCCCCCCCCCCCCCCCCCCC
NNN		BBBBBBBBBBBBBB	CCCCCCCCCCCCCCCCCCCCCCCC
NNNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNNN			CCCCCCCCCCCCCCCCCCCCCCCC
NNN			CCCCCCCCCCCCCCCCCCCCCCCC
			CCCCCCCCCCCCCCCCCCCCCCCC
			CCCCCCCCCCCCCCCCCCCCCCCC

Ready

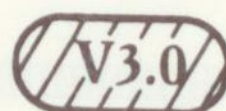
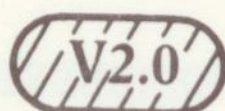
(Formato 2)

LINE[@][X₁,Y₁)-(X₂,Y₂),función [, [código de paleta][,{B|BF}]]

(Función)

La sentencia LINE en el formato 2 dibuja una línea o un rectángulo utilizando puntos.

(Versiones)



(Descripción)

- (X₁,Y₁) y (X₂,Y₂) son las coordenadas gráficas de cualquier par de puntos de la pantalla. Se puede dibujar o borrar una línea uniendo dichos puntos.
- La función se puede especificar como PSET, PRESET, AND, OR, o XOR. PSET hace que la línea aparezca en el color especificado, mientras que PRESET hace que la línea desaparezca. Con AND, OR, o XOR se realiza la operación lógica entre el código de paleta determinado para el color especificado y el color actual del punto; se usa el color resultante para dibujar la línea.
- Especificando B, se visualiza un rectángulo con ambos puntos en las esquinas diagonalmente opuestas. Especificando BF, se rellena el interior del rectángulo con el mismo color.
- Si se omite el código de paleta, se asume el color de primer plano utilizado en la última sentencia COLOR ejecutada.

(Ejemplos)

```
10 CLS
20 X=100:Y=50
30 LINE@ (X,Y)-200,Y,PSET
40 LINE@ -(300,100),PSET,2,B
50 LINE@ (320,50)-(500,150),PSET,4,BF
60 LINE@ (380,20)-(520,130),XOR,2,BF
70 LINE@ (10,60)-(180,100),PSET,5
80 STOP
90 LINE@ (10,60)-(180,100),PRESET
100 LINE@ (380,20)-(520,130),XOR,4,BF
```

Las líneas 50 y 60 dibujan unos rectángulos superpuestos. La sentencia STOP en la línea 80 detiene la ejecución. Al introducir el comando CONT, para continuar la ejecución, la línea que une los puntos (10,60) y (180,130) desaparece, y el color del rectángulo superior cambia.

3.4.10. CONNECT (conectar)

(Función)

La sentencia CONNECT une, mediante una línea recta, los puntos con las coordenadas especificadas.

(Formato)

CONNECT(X_1, Y_1)-(X_2, Y_2)[-(X_3, Y_3)]...
[, [código de paleta][, función]]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia conecta n puntos, cada uno con el siguiente, en el orden indicado dibujando líneas rectas.
- El código de paleta indica el color de las líneas de conexión. Si se omite, se usa el último color de primer plano utilizado en la última sentencia COLOR ejecutada.
- La función puede especificarse como PSET, PRESET, AND, OR o XOR.

(Ejemplos)

5 CLS

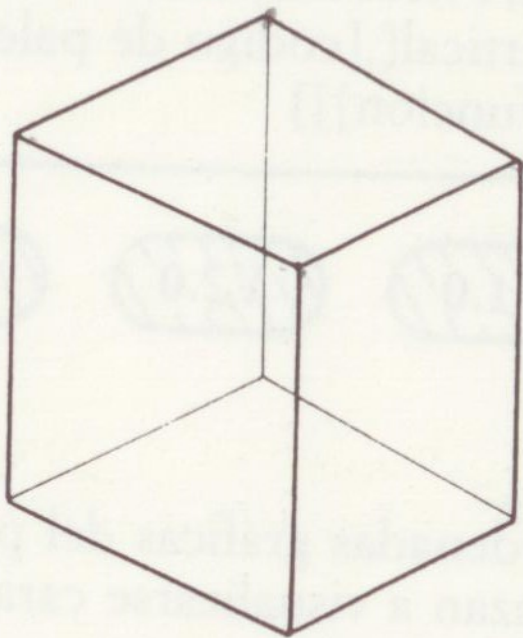
10 CONNECT (320,30)-(400,50)-(330,70)-
(240,50)-(320,30)-(320,90)-(400,110)-
(330,130)-(240,110)-(320,90),5,PSET

20 CONNECT (400,50)-(400,110),5,PSET

30 CONNECT (330,70)-(330,130),5,PSET

40 CONNECT (240,50)-(240,110),5,PSET

50 END



3.4.11. SYMBOL (símbolo)

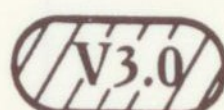
(Función)

La sentencia SYMBOL visualiza en pantalla una cadena de una longitud especificada en la posición y con el ángulo indicado.

(Formato)

SYMBOL (X,Y),cadena,tamaño horizontal, tamaño vertical[, [código de paleta][, código de ángulo] [,función]]

(Versiones)



(Descripción)

- (x,y) son las coordenadas gráficas del punto superior izquierdo en el cual comienzan a visualizarse caracteres.
- Los tamaños horizontal y vertical se aplican para ampliar el carácter dado, y cada carácter se visualiza con un patrón de (tamaño horizontal x 8) x (tamaño vertical x 8) puntos.
- El código de ángulo indica el ángulo en que gira cada carácter al visualizarlo. El valor por defecto es 0.

0: Normal
1: 90 grados en el sentido contrario a las agujas del reloj
2: 180 grados en el sentido contrario a las agujas del reloj
3: 270 grados en el sentido contrario a las agujas del reloj
- Si se omite el código de paleta, se utiliza el último color de primer plano utilizado en la última sentencia COLOR ejecutada.
- La función puede especificarse como PSET, PRESET, AND, OR, XOR o NOT.

- La sentencia SYMBOL no afecta más que a los puntos que se utilizan para visualizar los caracteres.

(Ejemplos)

5 CLS

10 SYMBOL (316,93), "A", 1, 1

20 SYMBOL (323,108), "A", 1, 1, 7, 2

30 SYMBOL (331,97), "A", 1, 1, 7, 3

40 SYMBOL (308,104), "A", 1, 1, 7, 1

50 SYMBOL (168,80), "A", 13, 10, 2

60 SYMBOL (224,50), "O", 24, 18, 4

3.4.12. GET @(obtener@)

(Formato 1)

GET@ (X₁,Y₁)-(X₂,Y₂),nombre de matriz

(Función)

La sentencia GET en el formato 1 lee caracteres de pantalla y los guarda en una matriz.

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- (X₁,Y₁) y (X₂,Y₂) son coordenadas de caracteres. La sentencia lee los caracteres que hay dentro de un rectángulo que tiene como esquinas diagonalmente opuestas las posiciones (X₁,Y₁) y (X₂,Y₂).
- La dimensión de la matriz que se debe reservar se calcula del siguiente modo:

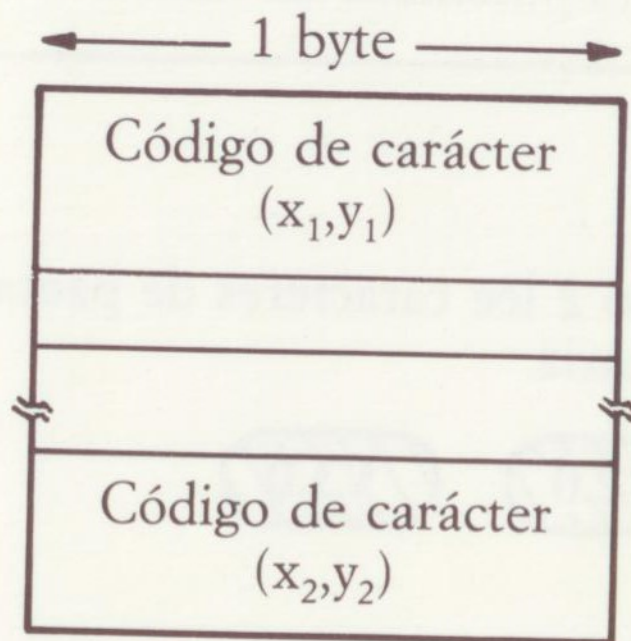
Tamaño de la matriz = INT((área expresada como número de caracteres + a-1)/a)

Donde "a" representa el tamaño de un elemento de la matriz.

a = 2: Tipo entero
4: Simple precisión
8: Doble precisión

Una matriz cadena produce un error "Illegal Function Call" (llamada ilegal a la función).

- Los códigos de los caracteres de la pantalla se guardan en la matriz del siguiente modo:



La carga a la matriz se realiza de izquierda a derecha y de arriba abajo en la pantalla.

(Ejemplos)

```

10 CLS
20 LOCATE 10,5:PRINT "F-BASIC"
30 DIM A%(INT((2+2-1)/2))
40 DIM B%(INT((5+2-1)/2))
50 GET@ (10,5)-(11,5),A%
60 GET@ (12,5)-(16,5),B%
70 PUT@ (15,8)-(15,9),A%
80 PUT@ (17,11)-(19,12),B%
90 LOCATE 0,21
100 END

```

F-BASIC

F

-

BAS
IC

El programa carga en una matriz los caracteres de la pantalla, y después los visualiza en una posición diferente con la sentencia PUT@.

(Formato 2)

GET@A (X₁,Y₁)-(X₂,Y₂),nombre de matriz

(Función)

La sentencia GET@ en el formato 2 lee caracteres de pantalla y sus atributos y los guarda en una matriz.

(Versiones)

V1.0

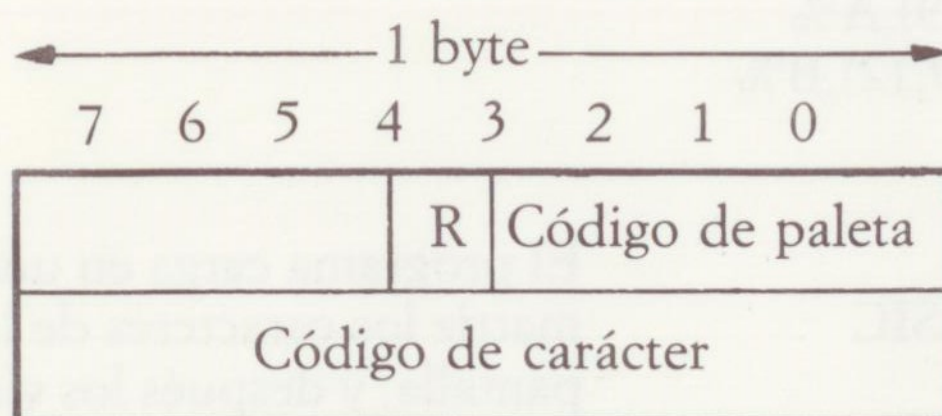
V2.0

V3.0

(Descripción)

- La sentencia GET@A lee los códigos de los caracteres y los códigos de paleta.
- El tamaño de la matriz debe ser el doble que con el formato 1.

Cada carácter ocupa 2 bytes. El primer byte es para el atributo del carácter; el segundo byte es para el código del carácter. Los dos bytes se almacenan con el siguiente formato:



R 0: normal
1: inverso

(Ejemplos)

```
10 CLS
20 LOCATE 10,5
30 COLOR 2:PRINT "ABC";
40 COLOR 4:PRINT "DEF";
45 COLOR 1:PRINT "GHI"
50 DIM A%(INT((18+2-1)/2))
60 GET@ A (10,5)-(18,5),A%
70 PUT @ A (13,9)-(21,9),A%
80 COLOR 7,0
100 END
Ready
RUN
```

ABCDEFGHII

(Formato 3)

GET@A (X₁,Y₁)-(X₂,Y₂),nombre de matriz,G
[,código de paleta[,código de paleta]..]

(Función)

La sentencia GET@ en el formato 3 lee conjuntos de puntos de pantalla y los guarda en una matriz.

(Versiones)

V1.0

V2.0

V3.0

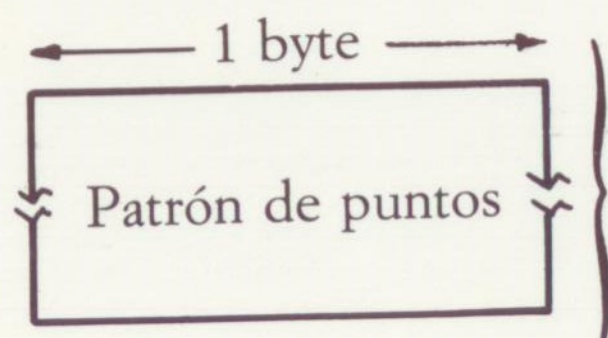
(Descripción)

- (x₁,y₁) y (x₂,y₂) son coordenadas gráficas. La sentencia lee los caracteres que hay dentro de un rectángulo que tiene como esquinas diagonalmente opuestas las posiciones (x₁,y₁) y (x₂,y₂).
- Se pueden especificar hasta 8 códigos de paleta. Si el color de un punto coincide con uno de los códigos de paleta especificados, el bit correspondiente de la matriz se pone en 1. Si no se especifica ningún código de paleta y el punto se está visualizando en un color diferente del de fondo, el bit correspondiente de la matriz también se pone en 1.
- La dimensión de la matriz que se debe reservar se calcula del siguiente modo:

Tamaño de la matriz = $\text{INT}(\text{INT}((\text{número total de puntos} + 7)/8 + a - 1)/a)$

donde a representa el tamaño de un elemento de la matriz.

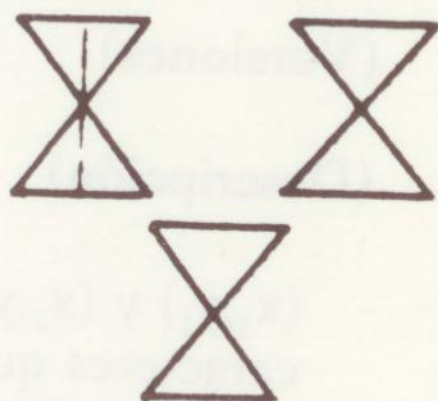
- Los datos se almacenan en la matriz del siguiente modo:



INT((Número total de puntos + 7)/8) bytes. Los unos corresponden a los puntos visualizados en la pantalla.

(Ejemplos)

```
10 CLS
15 LINE @(60,5)-(60,15),PSET,2
20 CONNECT (50,5)-(70,5)-(50,15)-(70,15)-(50,5),7,PSET
30 DIM A%(INT((INT((231+7)/8)+2-1)/2))
35 DIM B%(INT((INT((231+7)/8)+2-1)/2))
40 GET@(50,5)-(70,15),A%,G,7
41 GET@(50,5)-(70,15),B%,G,2
50 PUT@(100,5)-(120,15),A%,PSET,7
60 PUT@(80,20)-(100,30),A%,PSET,4
```



(Formato 4)

GET@ A (X₁,Y₁)-(X₂,Y₂),nombre de matriz,G

(Función)

La sentencia GET@ en el formato 4 lee de pantalla conjuntos de puntos y su color, y los guarda en una matriz.

(Versiones)

V1.0

V2.0

V3.0

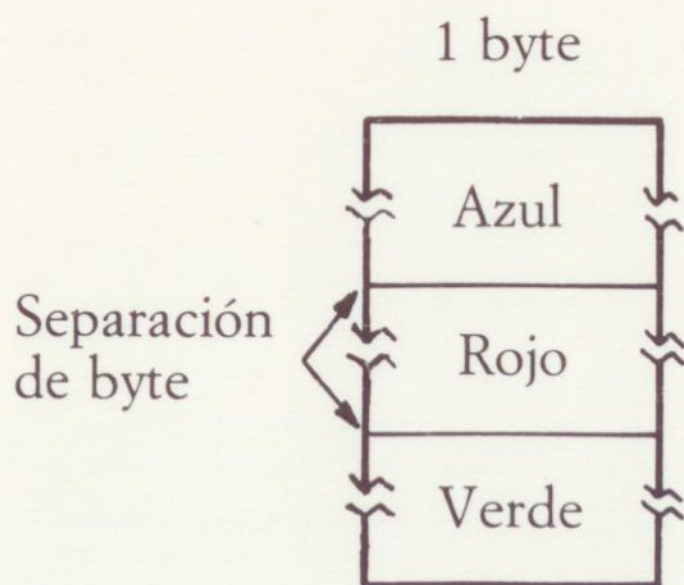
(Descripción)

- (x₁,y₁) y (x₂,y₂) son coordenadas gráficas. La sentencia lee los caracteres que hay dentro de un rectángulo que tiene como esquinas diagonalmente opuestas las posiciones (x₁,y₁) y (x₂,y₂).
- La dimensión de la matriz que se debe reservar se calcula del siguiente modo:

Tamaño de la matriz = INT(INT((número total de puntos + 7)/8 x 3 + a-1)/a))

donde a representa el tamaño de un elemento de la matriz.

Los datos se almacenan en la matriz del siguiente modo:



INT((Número total de puntos + 7)/8) bytes.

El color de cada punto se representa mediante la continuación de rojo, verde y azul.

(Ejemplos)

```
10 CLS
15 COLOR 7,1
16 CLS
20 LINE(200,20)-(240,40),PSET,2,BF
30 LINE(200,20)-(220,29),XOR,1,BF
40 LINE(240,40)-(220,31),XOR,4,BF
50 LINE(220,20)-(240,30),XOR,6,BF
60 DIM A%(INT((INT((861+7)/8)*3+2-1)/2))
70 GET@ A(200,20)-(240,40),A%,G
80 PUT@ A(100,70)-(140,90),A%,NOT
90 PUT@ A(150,70)-(190,90),A%,PSET
100 PUT@ A(200,70)-(240,90),A%,OR
110 PUT@ A(250,70)-(290,90),A%,XOR
120 PUT@ A(300,70)-(340,90),A%,AND
130 LOCATE 0,20
140 END
```


3.4.13. PUT@ (poner @)

(Formato 1)

PUT@ (X₁,Y₁)-(X₂,Y₂),nombre de matriz[,código de paleta]

(Versiones)

V1.0

V2.0

V3.0

(Función)

La sentencia PUT@ en el formato 1 visualiza los caracteres leídos con una sentencia GET@ (formato1), en una posición especificada de la pantalla.

(Descripción)

- El formato de la matriz y los datos deben ser iguales a los descritos en el formato 1 de la sentencia GET.
- El código de paleta indica el color de los caracteres visualizados. Si se omite este código, se usa el color de primer plano utilizado en la última sentencia COLOR ejecutada.

(Ejemplos)

Ver los ejemplos en la Sección 3.4.12, formato 1.

(Formato 2)

PUT@A (X₁,Y₁)-(X₂,Y₂),nombre de matriz

(Función)

La sentencia PUT@ A en el formato 2 visualiza los caracteres leídos con una sentencia GET@ A (formato 2), en una posición especificada de la pantalla.

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El formato de la matriz y los datos han de ser los mismos descritos en el formato 2 de la sentencia GET@A.

(Ejemplos)

Ver los ejemplos en la Sección 3.4.12, formato 2.

(Formato 3)

PUT@ (X₁,Y₁)-(X₂,Y₂),nombre de matriz,función
[,código de paleta]

(Función)

La sentencia PUT@ en el formato 3 visualiza los puntos gráficos leídos con una sentencia GET@ (formato 3), en una posición especificada de la pantalla.

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El formato de la matriz y los datos han de ser iguales a los descritos en el formato 3 de la sentencia GET@.
- Como se ha descrito anteriormente, los bits de la matriz no proporcionan ninguna información de color; lo único que indican es si se visualiza o no un punto. Por lo tanto, ha de especificarse el código de paleta para indicar el color de visualización. Si se omite este código de paleta, se usa el color de primer plano utilizado en la última sentencia COLOR ejecutada.
- La función se puede especificar como PSET, PRESET, AND, OR, XOR o NOT. PSET hace que todos los puntos cuyo bit en la matriz sea 1, aparezcan en el color especificado, y que los otros puntos (cuyo bit es 0) permanezcan sin cambiar. PRESET hace que todos los puntos con bit igual a 1 cambien al color del fondo. NOT hace que todos los puntos con bit igual a 0 se visualicen en el color especificado. Si se especifica AND, OR ó XOR, para cada bit igual a 1 en la matriz, se realiza una operación lógica entre el color especificado de paleta y el color de paleta actual del punto; el resultado se usa como color de visualización. Los puntos con bit igual a 0 no se modifican.

(Ejemplos)

Ver los ejemplos en la Sección 3.4.12, formato 3.

(Formato 4)

PUT @A (X₁,Y₁)-(X₂,Y₂),nombre de matriz,función

(Función)

La sentencia PUT@A en el formato 4 visualiza los datos gráficos leídos con la sentencia GET@A (formato 4) en la posición de pantalla especificada.

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El formato de la matriz y los datos han de ser iguales a los descritos en el formato 4 de la sentencia GET@A.
- La función se puede especificar como AND, OR, XOR, NOT o PSET. Si se especifica AND, OR o XOR, se realiza la operación lógica entre los bits R(rojo)-G(verde)-B(azul) del punto que está actualmente en la pantalla y los bits R-G-B en la matriz. El resultado se visualiza en la pantalla. NOT invierte los tres bits rojo, verde y azul de la matriz y visualiza los colores resultantes. Esto produce una visualización de colores complementarios a los indicados en la matriz. PSET visualiza los datos de la matriz en la pantalla sin cambiarlos.

(Ejemplos)

Ver los ejemplos de la Sección 3.4.12 formato 4.

3.4.14. CIRCLE (círculo)

(Función)

La sentencia CIRCLE dibuja círculos, elipses y arcos de elipse con centro en una posición especificada de la pantalla.

(Formato)

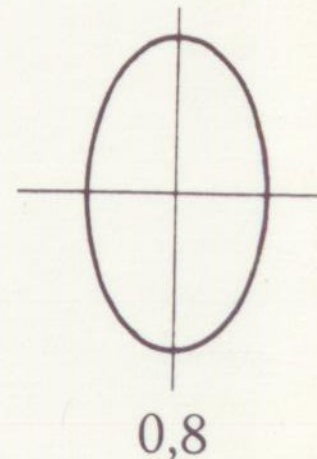
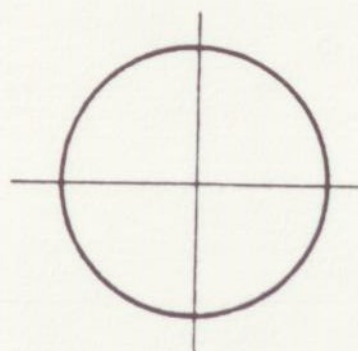
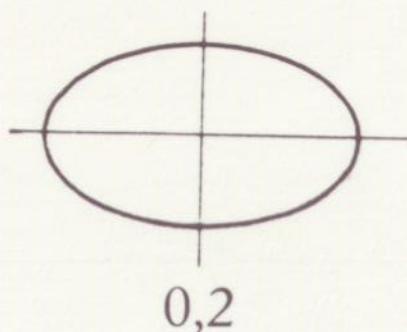
```
CIRCLE[@] (X,Y)
          radio[, [código de paleta][, [relación]
          [, [posición de comienzo][, [posición final]
          [, [{F|N}][, función]]]]]
```

(Versiones)

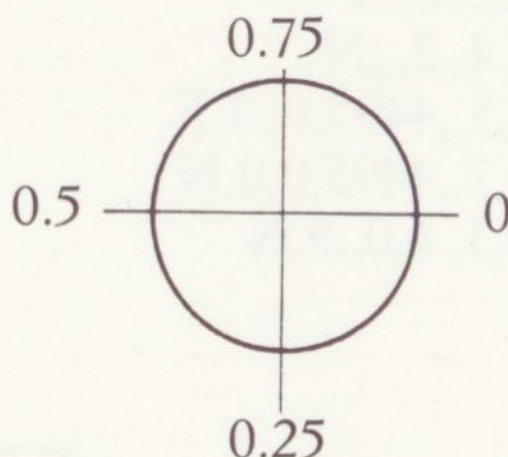


(Descripción)

- (x,y) indica la posición gráfica del centro.
- Especifica el radio en el eje X mediante un número de puntos.
- El código de paleta indica el color con el cual se dibuja el círculo (elipse) o el arco. Si se omite, se usa el color de primer plano utilizado en la última sentencia COLOR ejecutada.
- La relación es el cociente entre el número de puntos en X y el número de puntos en Y, y debe estar entre 0 y 1. Si se omite la relación, se asume 0,4495 y se dibuja un círculo.



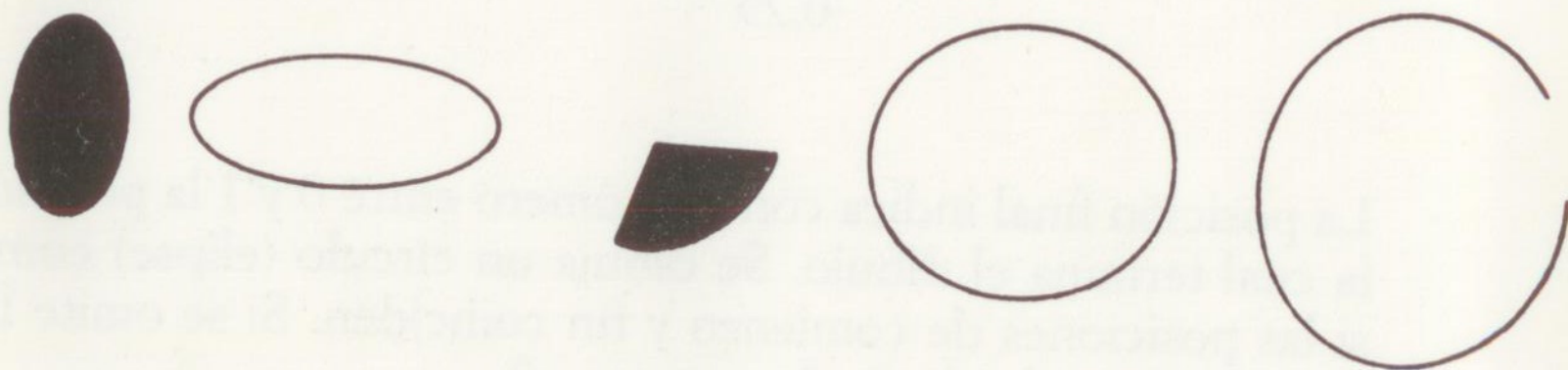
- La posición de comienzo indica con un número entre 0 y 1, la posición en el perímetro en la cual se empieza a dibujar.



- La posición final indica con un número entre 0 y 1 la posición en la cual termina el dibujo. Se dibuja un círculo (elipse) completo si las posiciones de comienzo y fin coinciden. Si se omite la posición inicial o la final, se asume 0.
- Si se especifica F, se rellena el interior del círculo (elipse). Si se especifica N se dibuja solamente la circunferencia. Si no se especifica ninguno, se asume N.
- La función se puede especificar como PSET, PRESET, AND, OR o XOR. Si no se especifica ninguna, se asume PSET.
- El valor mínimo reconocible para la posición inicial o la final es 0,0156. Por consiguiente, el arco circular mínimo es un 64avo de un círculo completo. No se pueden dibujar arcos más cortos.
- Si se especifica F, se rellena el interior del círculo dibujando líneas desde el centro hasta el perímetro. La función PSET, PRESET, AND, OR o XOR controlan la manera en que se dibuja la línea. Por ejemplo, si se especifica XOR, se obtiene un círculo rayado.

(Ejemplos)

```
10 CLS
20 CIRCLE(60,90),20,2,.90,,,F
30 CIRCLE(150,90),50,4,.2,,,N
40 CIRCLE(280,90),40,5,.4495,0,.3,F
50 CIRCLE(400,90),50,7,.4495,0,0,N
60 CIRCLE(530,90),50,3,.6,0,.9,N
70 END
```



Nota:

El espaciado de los puntos en las direcciones X e Y es diferente en la pantalla y en la impresora.

3.4.15. GCURSOR (cursor gráfico)

(Función)

La sentencia GCURSOR lee las coordenadas del cursor gráfico en la pantalla.

(Formato)

GCURSOR (x,y),(nombre de variable 1,nombre de variable 2)
[, (nombre de variable 3,nombre de variable 4)...]
[, código de paleta]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- (x,y) son las posiciones en las cuales se posiciona inicialmente el cursor. El cursor gráfico aparece como un signo más (+) en la posición especificada y en el color especificado por el código de paleta.
- (nombre de variable 1, nombre de variable 2) hace referencia a las áreas de memoria en las cuales se guarda la posición leída del cursor gráfico. Las coordenadas se leen cuando se pulsa la tecla RETURN.
- Antes de leer cada par de coordenadas, el usuario puede mover el cursor gráfico (+) a la posición deseada mediante las teclas de movimiento del cursor. Cuando se han leído las coordenadas de todos los puntos, el cursor gráfico desaparece.

El modo en que avanza el cursor es el siguiente:

- * Si se pulsa una tecla de movimiento del cursor, éste se mueve un punto en la dirección especificada por la tecla.

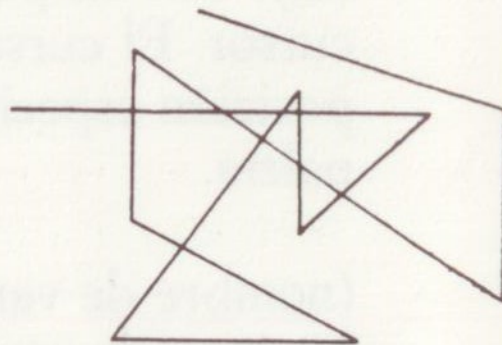
- * Si se pulsa una tecla de movimiento del cursor después de una tecla numérica (0 a 9), el cursor se mueve dicho número de puntos.
- * Si se pulsa la tecla SHIFT al mismo tiempo que una tecla de movimiento del cursor, éste se mueve veinte puntos.
- Si se omite el código de paleta, se usa el color de primer plano utilizado en la última sentencia COLOR ejecutada.
- Se pueden leer hasta 10 coordenadas.

(Ejemplos)

```

10 CLS
20 GCURSOR (320,100),(X,Y)
30 LINE@ (0,0)-(X,Y),PRESET
35 FOR I=1 TO 10
40 GCURSOR (X,Y),(X,Y)
50 LINE@-(X,Y),PSET
60 NEXT I
70 END

```



3.4.16. PAINT (pintar)

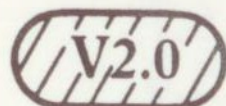
(Función)

La sentencia PAINT rellena un área rodeada por unos límites de colores especificados.

(Formato)

PAINT(posición horizontal,posición vertical)[,[código de paleta]
[,color límite 1[,color límite 2]...]]

(Versiones)



(Descripción)

- (posición horizontal, posición vertical) son las coordenadas del punto en el que se comienza a rellenar.
- El código de paleta indica el color con el que se rellena. Si se omite el código de paleta, se usa el color de primer plano utilizado en la última sentencia COLOR ejecutada.
- Se puede utilizar uno o más colores límite. Los colores límite se especifican por sus códigos de paleta. El código de paleta que precede a los colores de los límites es también un color de límite.
- Si no se especifica color de límite, el color de paleta especificado es el único color límite.
- Si la región a rellenar está dividida en muchas secciones pequeñas, podría ocurrir que algunas se quedasen sin pintar debido a limitaciones del área de trabajo.

(Ejemplos)

```
10 CLS
15 C=0
15 Y=0
20 FOR X=0 TO 200 STEP 25
30 LINE(320-X,100-Y)-(320+X,100+Y),PSET,C,B
40 Y=Y+10
50 C=C+1
60 NEXT X
70 PAINT(320,109),2,1
80 PAINT(380,129),6,2,3
90 PAINT(430,149),1,4,5
100 PAINT(480,169),4,6,7
110 PAINT(455,159),3,5,6
120 PAINT(405,139),7,3,4
130 PAINT(355,119),5,1,2
140 END
```

El bucle FOR NEXT entre la línea 20 y la línea 60 dibuja 7 cajas concéntricas, y las sentencias PAINT entre la línea 70 y la línea 130 pinta el dibujo en colores concéntricos.

3.5. Funciones musicales

3.5.1. PLAY (componer)

(Función)

La sentencia PLAY emite notas musicales.

(Formato)

PLAY "MML"[,"MML"[,"MML"]]

(Versiones)

V3.0

(Descripción)

La sentencia PLAY, provoca la emisión de notas musicales utilizando los comandos codificados en el Music Macro Language (MML) o macro lenguaje de música. Los comandos MML especifican tono, duración, tiempo, y otros parámetros. Se pueden obtener acordes de hasta tres notas, separando los comandos MML con comas. Se dispone de los siguientes comandos:

(1) Tono (A, B, C, D, E, F, G)

Los comandos A al G especifican tonos: C = do, D = re, E = mi, F = fa, G = sol, A = la, y B = si. Se pueden añadir a continuación de las letras el signo de sostenido (# ó +) o el signo de bemol (-) para obtener semitonos.

(2) Silencio (Rest)

El comando R especifica un silencio.

(3) Duración de la nota (Ln)

El comando Ln especifica la duración relativa de las notas. El número n debe ser un entero entre 1 y 64. Representa los inversos de la duración del sonido. De modo que 1 es una nota redonda, 2 es una nota blanca, y así sucesivamente. Una vez introducido el comando Ln, todas las notas y silencios tienen la misma duración a no ser que se agregue una n diferente al comando de tono o de silencio. Agregando una n al comando de tono o de silencio se controla la duración relativa de la nota.

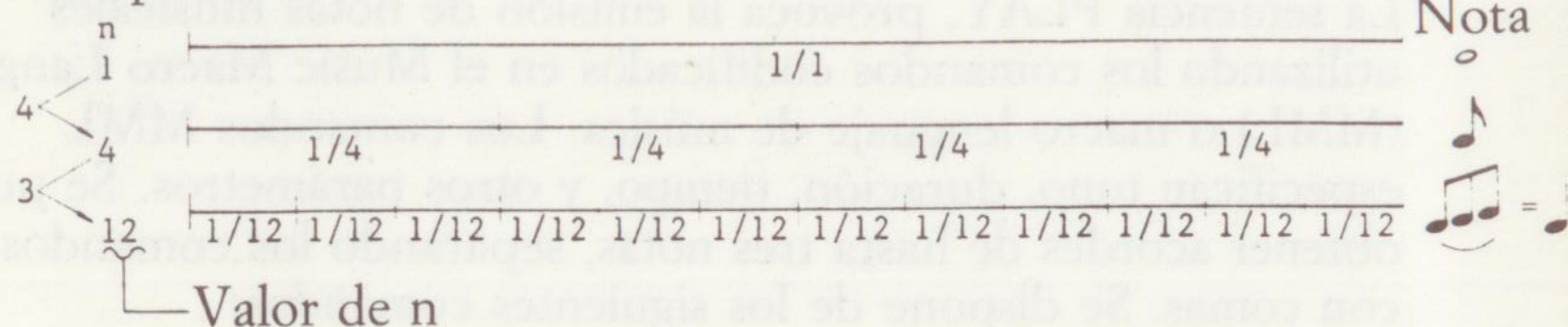
(Ejemplos) Nota negra Do. L4C ó C4

Semicorchea Fa sostenido: L16F#, F#16, ó F+16

Redonda Sol bemol: L1G- ó G-1

Normalmente, la regla inversa da el valor correcto de n, pero obsérvese que para tres notas consecutivas, n se determina del siguiente modo:

Multiplicador



Agregando un punto (.) al comando de tono con designación de duración (An a Gn) alarga la duración de la nota en un factor de 3/2. Por ejemplo, A4.. da una duración de 9/4 la duración de una negra, que corresponde a n=9.

Tabla de duraciones

Nota			Silencio	
Símbolo	Nombre	n	Símbolo	Nombre
	Nota redonda	1		Silencio redonda
	Blanca	2		Silencio blanca
	Negra	4		Silencio negra
	Corchea	8		Silencio corchea
	Semicorchea	16		Silencio semicorchea

(4) Octava (On)

El comando O especifica la octava. n debe ser un entero entre 1 y 8. El tono La estándar (frecuencia 440 Hz.) está en la octava cuatro (O4).

(5) Escala dodecafónica (Nn)

Aunque normalmente las notas se representan con los comandos A al G, también se pueden especificar con el comando Nn, en el que n es un entero entre 1 y 96. N1 es el Do en la octava 1 (O1C), y desde aquí el tono crece en intervalos de semitonos de modo que N96 es el Si en la octava 8 (O8B).

Tabla de correspondencia

	01		02		03		04		05		06		07		08	
C	1	2	13	14	25	26	37	38	49	50	61	62	73	74	85	86
D	3	4	15	16	27	28	39	40	51	52	63	64	75	76	87	88
E	5	/	17	/	29	/	41	/	53	/	65	/	77	/	89	/
F	6	7	18	19	30	31	42	43	54	55	66	67	78	79	90	91
G	8	9	20	21	32	33	44	45	56	57	68	69	80	81	92	93
A	10	11	22	23	34	35	46	47	58	59	70	71	82	83	94	95
B	12	/	24	/	36	/	48	/	60	/	72	/	84	/	96	/

En esta tabla, cada octava tiene dos columnas: la columna de la izquierda representa la nota normal; la columna de la derecha es un semitono más aguda.

Ejemplo 1: Valor Nn para O4A#

Puesto que O4 en la fila de comandos A corresponde a 46 o 47 y la nota tiene un signo de sostenido (#), O4A# corresponde a N47.

Ejemplo 2: Valor Nn para O3B-

Puesto que B- es un semitono más bajo que B, es un semitono más agudo que A, de modo que O3B- corresponde a N35.

(6) Tiempo (Tn)

El comando Tn especifica el tiempo. El número n debe ser un entero entre 32 y 255, y representa el número de negras por minuto.

(7) Volumen (V_n)

El comando V_n especifica el volumen de una parte determinada. El número n debe ser un entero entre 0 y 15. V_{15} da el volumen máximo; V_0 no da ningún sonido para esa parte.

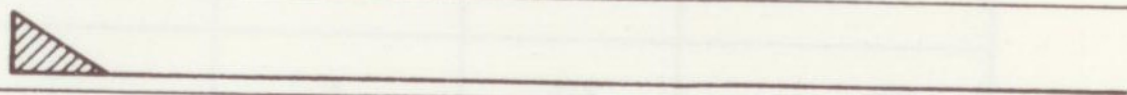
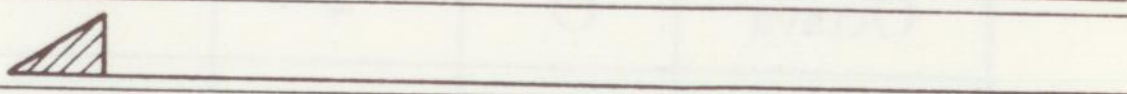
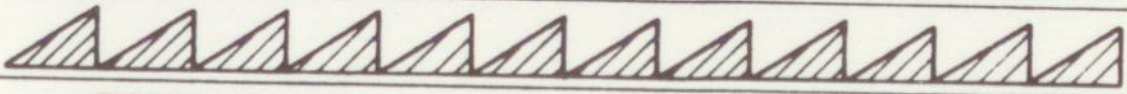
(8) Tipo de envolvente (S_n)

El comando S_n especifica un tipo de envolvente. El número n debe ser un entero entre 0 y 15. Cuando se introduce este comando, el comando V queda invalidado, puesto que el volumen se controla con el tipo de envolvente. Después de liberar este comando, el comando V vuelve a tomar control del volumen.

(9) Frecuencia de la envolvente (M_n)

El comando M debe usarse con el comando S . El número n debe ser un entero entre 0 y 65535. Determina la frecuencia de la envolvente. A continuación se indican los tipos de envolvente y la fórmula para calcular la frecuencia:

Correspondencia entre el tipo de envolvente y n en el comando S_n

n	Tipo de envolvente (El eje vertical indica volumen; (El eje horizontal indica tiempo.)
0 a 3 y 9	
0 a 7 y 15	
8	
10	
11	
12	
13	
14	

→ T ←

Relación entre el periodo T y el valor n en el comando Mn.

$$T = \frac{256.D}{f} \quad (f=1.2288 \text{ MHz})$$

- (10) Especificación indirecta (=nombre de variable;)

En la descripción anterior, n era constante, pero también se puede especificar una variable.

Ejemplo: I=10:PLAY"O2L=I;" es equivalente a PLAY"O2L10".

- (11) Valores por defecto

Algunos de los comandos descritos anteriormente tienen valores por defecto. La siguiente tabla muestra dichos valores.

Comando	Símbolo	Valor por defecto	Comentario
Duración	L	4	La duración en los comandos A al G y R se determina con el valor especificado en el comando L.
Octava	O	4	
Tiempo	T	120	
Volumen	V	8	

Notas:

- (1) El comando PLAY no se puede utilizar junto con comandos relacionados con el cassette de audio. Si se ejecuta un comando PLAY con "CAS0:" abierto, se produce un error "Device In Use" (dispositivo ocupado).
- (2) El comando PLAY se detiene cuando:
 - Se pulsa la tecla BREAK:
 - Se pulsa CTRL-X o CTRL-C cuando se espera una entrada.
 - Se ejecuta un comando relacionado con el cassette (excepto MOTOR).
 - Se ejecuta un comando RUN.
 - Se visualiza ABORT.
- (3) El comando PLAY se puede programar para obtener música de fondo mientras se están ejecutando otros procesos, puesto que se almacenan en memoria 3 voces y 18 sonidos, y las melodías pueden generarse con interrupciones.
- (4) Mientras el comando PLAY proporciona música de fondo, también se puede ejecutar el comando SOUND. Sin embargo, ya que el comando SOUND vuelve a escribir en los registros del generador de sonidos programable (PSG), la música resultará mezclada con/o sustituida por ruido. El PSG vuelve a su estado inicial en las condiciones descritas en (2).

3.5.2. SOUND (sonido)

(Función)

La sentencia SOUND controla directamente el generador de sonido programable (PSG).

(Formato)

SOUND PSG número de registro, dato

(Versiones)

V3.0

(Descripción)

La sentencia SOUND controla el generador de sonido programable escribiendo directamente el contenido de los registros del PSG. Los registros del PSG están comprendidos entre 0 y 13, y se pueden escribir datos entre 0 y 255. Las Figuras 1 y 2 muestran las funciones de cada registro. El FM-7 tiene un generador de sonido programable AY-3-8913. La Figura 1 muestra el diagrama de bloques. El generador de sonido programable está formado por tres osciladores, un generador de ruido, un generador de envolvente, un mezclador y tres atenuadores; y funciona como un simple sintetizador.

La Figura 2 muestra los 14 registros del PSG. Los registros R0 a R5 determinan la frecuencia de oscilación de los tres osciladores. R0 y R1, R2 y R3, y R4 y R5 están emparejados, y son válidos sus 12 bits menos significativos. La frecuencia de oscilación se calcula del siguiente modo:

$$f = \frac{f_{\text{reloj}}}{16 \times D}$$

$$D = \frac{f_{\text{reloj}}}{16 \times f}$$

donde D representa los datos que se deben escribir, y $f_{\text{reloj}} = 1.2288$ MHz

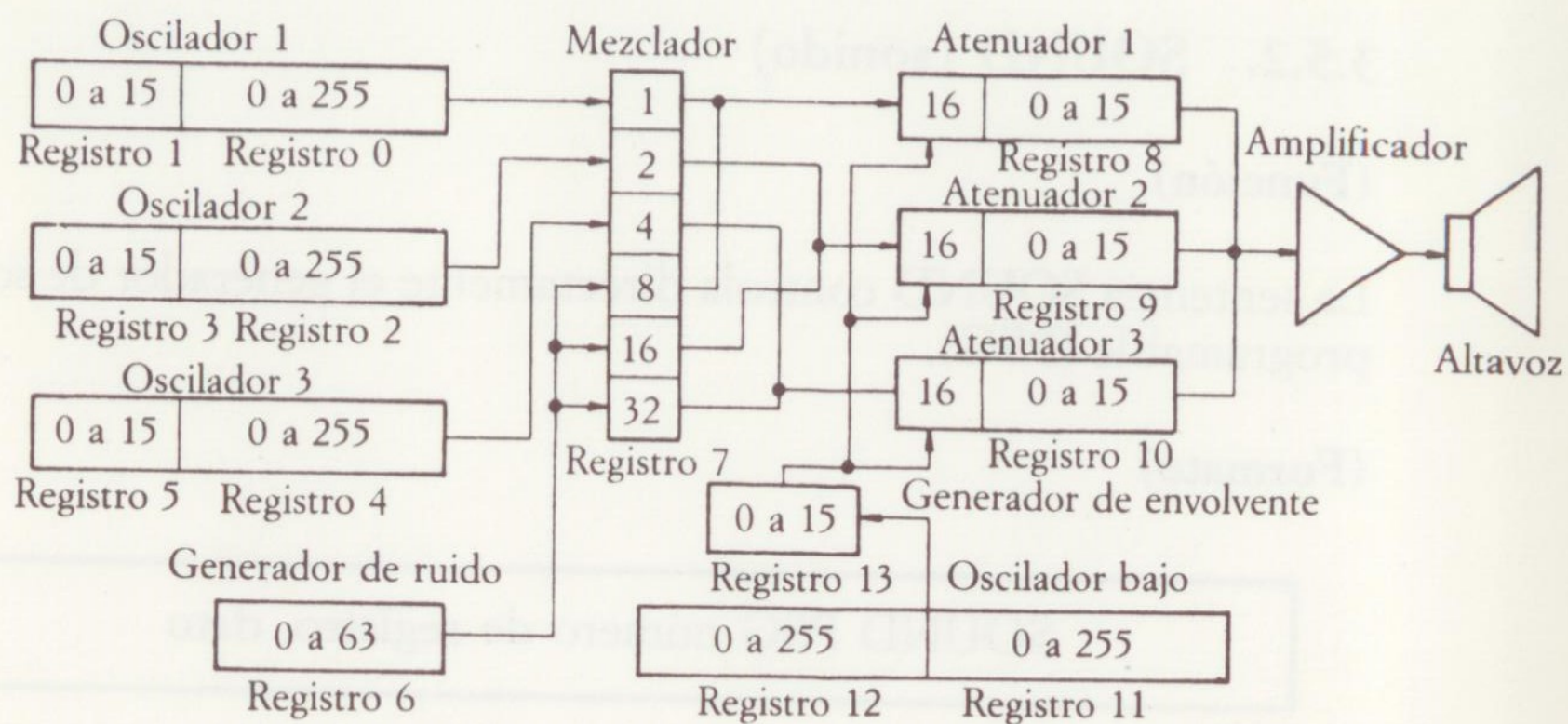


Fig. 1. Diagrama de bloques del PSG

Bit Registro	Operación	7	6	5	4	3	2	1	0	
R0	Tono del canal A	8 bits bajos								
R1						4 bits altos				
R2	Tono del canal B	8 bits bajos								
R3						4 bits altos				
R4	Tono del canal C	8 bits bajos								
R5						4 bits altos				
R6	Frecuencia de ruido	dato de 5 bits								
R7	Selección E/S				Ruido			Tono		
			C	B	A	C	B	A		
R8	Volumen canal A				dato 4 bits					
R9	Volumen canal B				dato 4 bits					
R10	Volumen canal C				dato 4 bits					
R11	Periodo de envolvente	8 bits bajos								
R12		8 bits altos								
R13	Forma de onda de la envolvente					C	A	A	H	

La escala musical se expresa con un dato de 12 bits

Frecuencia media del ruido

Cuando M=0, los 4 bits menos significativos controlan el volumen. Cuando M=1, se activa el generador de envolvente.

C=CONT
A=ATEN
A=ALT
H=HOLD

Fig. 2. Registros PSG

Para obtener el tono “La” (440Hz) utilizado para afinar instrumentos musicales, se escribe dato 349 en el registro deseado.

El registro R6 determina la frecuencia media de ruido. Sólo son válidos los 5 bits inferiores. La frecuencia puede calcularse del siguiente modo:

$$f = \frac{f_{\text{reloj}}}{16 \times D}$$

$$D = \frac{F_{\text{reloj}}}{16 \times f}$$

El registro R7 selecciona el modo de salida PSG. Para establecer el modo, se escriben ceros en los bits correspondientes.

Los 4 bits menos significativos de los registros R8 a R10 establecen el volumen. El volumen máximo se obtiene con el valor 15. Si el bit 4 se pone a 1, se activa el generador de envolvente y se ignora la especificación de volumen. (El volumen es controlado por el generador de envolvente).

Los registros R11 a R13 controlan el generador de envolvente. El tipo de envolvente puede seleccionarse de entre los tipos en la Figura 3. Escribiendo el código de forma de onda en el registro 13, se selecciona la forma de onda y, al mismo tiempo, se dispara el generador de envolvente. La longitud de la envolvente se especifica en los registros R11 y R12 como frecuencia de envolvente. La frecuencia de envolvente se calcula con la siguiente fórmula:

$$f = \frac{f_{\text{reloj}}}{256 \times D}$$

Utilizando el valor calculado de f , se deduce el tiempo E de la Figura 3 del siguiente modo:

$$\varepsilon = \frac{1}{f}$$

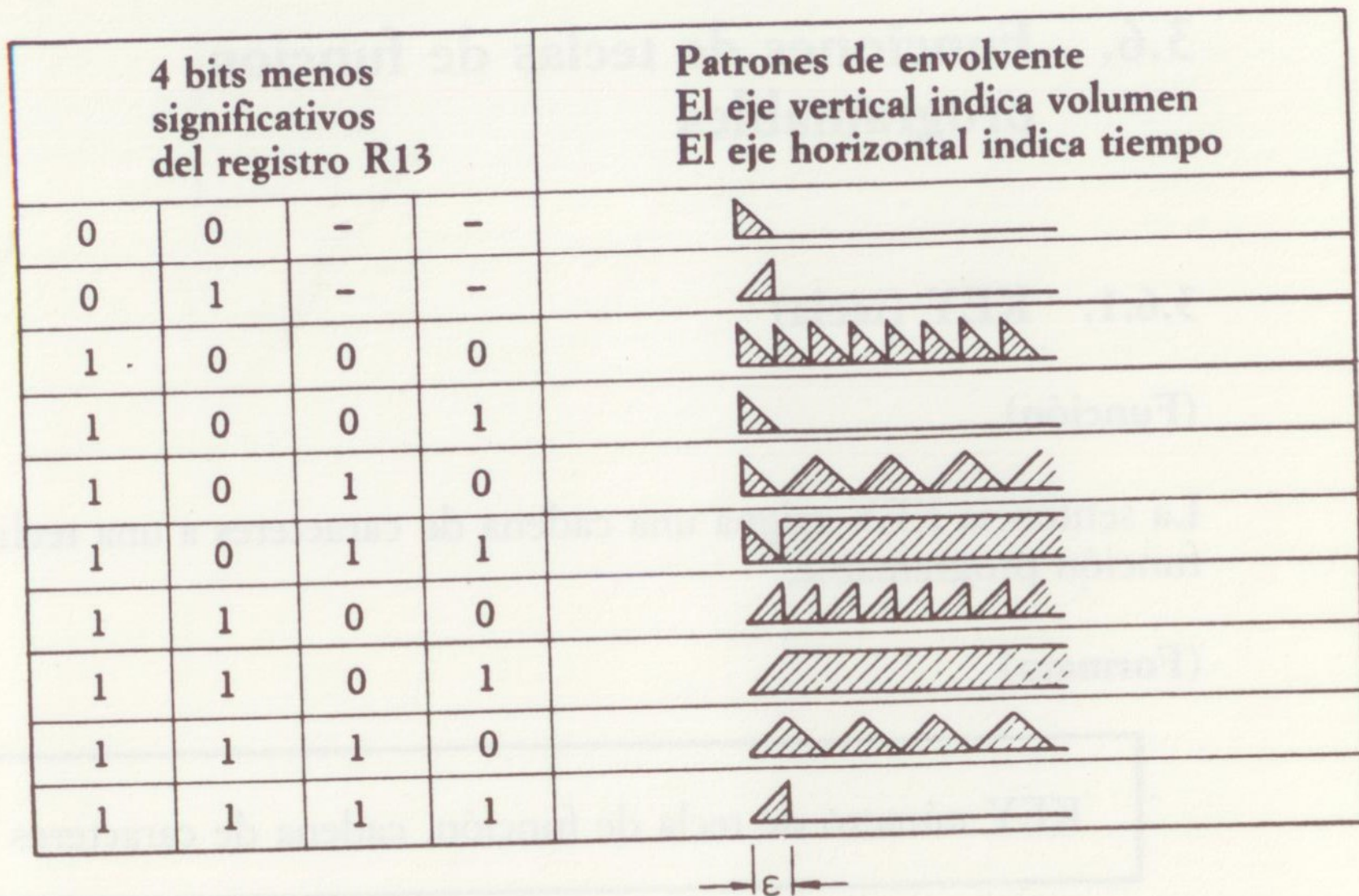


Fig. 3. Registro R13 y patrones de envolvente

3.6. Funciones de teclas de función programables

3.6.1. KEY (tecla)

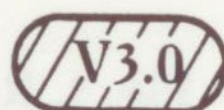
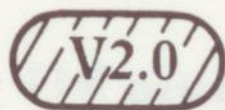
(Función)

La sentencia KEY asigna una cadena de caracteres a una tecla de función programable.

(Formato)

KEY número de tecla de función, cadena de caracteres

(Versiones)



(Descripción)

- El número de tecla de función corresponde al número de la tecla PF en el teclado; debe estar comprendido entre 1 y 10.
- La cadena de caracteres puede tener hasta 15 caracteres de longitud, incluidos los de control. Para incluir caracteres de control, se especifica una expresión de cadena formada por una cadena de caracteres, un signo más (+) y la función CHR\$.

(Ejemplos)

KEY 1,"LOCATE"

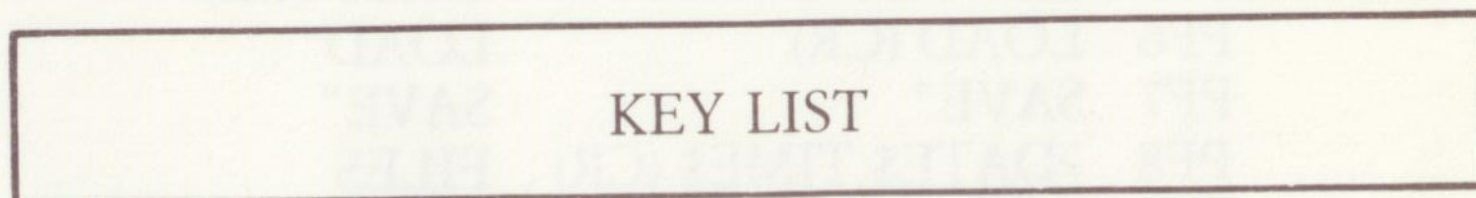
LOCATE se asigna a la tecla de función programable 1.

3.6.2. KEY LIST (lista de teclas)

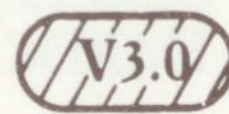
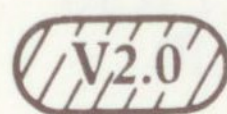
(Función)

La sentencia KEY LIST visualiza las cadenas de caracteres asignadas a las teclas de función programable.

(Formato)



(Versiones)



(Descripción)

- Las teclas de función programable se establecen con las siguientes cadenas de caracteres al encender el equipo y al hacer reset del sistema.

(1) Para V1.0 y V2.0:

PF1	AUTO_
PF2	LIST (CR)
PF3	RUN (CR)
PF4	CONT (CR)
PF5	LLIST (CR)
PF6	LOAD (CR)
PF7	SAVE"
PF8	?DATE\$,TIME\$ (CR)
PF9	SCREEN 7, 7 (CR)
PF10	HARDC (CR)

(2) Para V3.0:

	modo ROM	modo DISCO
PF1	AUTO_	AUTO_
PF2	LIST (CR)	LIST (CR)
PF3	RUN (CR)	RUN (CR)
PF4	CONT (CR)	CONT (CR)
PF5	LLIST (CR)	LLIST (CR)
PF6	LOAD (CR)	LOAD"
PF7	SAVE"	SAVE"
PF8	?DATE\$,TIME\$ (CR)	FILES
PF9	SCREEN7,7 (CR)	SCREEN7,7 (CR)
PF10	HARDC (CR)	HARDC (CR)

3.6.3. KEY(n) ON/OFF/STOP (tecla(n) activar/desactivar/aplazar)

(Función)

La sentencia KEY(n) ON/OFF/STOP permite, inhibe o aplaza las interrupciones desde las teclas de función.

(Formato)

`KEY(número de tecla de función) { ON | OFF | STOP }`

(Versiones)



(Descripción)

- KEY(n) ON permite las interrupciones producidas por la tecla de función n. Si se pulsa la tecla de función n después de la ejecución de esta sentencia, se llama a una rutina de proceso de la interrupción definida con la sentencia ON KEY(n) GOSUB.
- KEY(n) OFF inhibe las interrupciones producidas por la tecla de función n.
- KEY(n) STOP aplaza temporalmente el proceso de la interrupción producida por la tecla de función n. Es decir, si se pulsa la tecla de función n después de la ejecución de la sentencia KEY(n) STOP, no se llama inmediatamente a la rutina de proceso de la interrupción. No se la llama hasta que se ejecute una sentencia KEY(n) ON.
- La ejecución del comando RUN pone todas las teclas de función en OFF.
- No se puede introducir la cadena de caracteres asignada a la tecla de función mientras KEY ON esté activa; por lo tanto, se debe ejecutar un KEY OFF después del proceso de interrupción.

(Ejemplos)

```
1 ON KEY (1) GOSUB 60
5 KEY (1) ON
10 CLS
20 C=1
30 COLOR C
40 PRINT " ";
50 GOTO 40
60 BEEP
70 C=C+1
80 IF C=8 THEN C=1
90 RETURN 30
```

Las líneas 40 y 50 visualizan en pantalla un cuadrado en un color fijo. Pulsando la tecla de función 1, se suma 1 al código de color, por lo que éste cambia.

3.6.4. ON KEY(n) GOSUB (en tecla(n) ir a subrutina)

(Función)

La sentencia ON KEY(n) GOSUB define una rutina de interrupción de una tecla de función.

(Formato)

ON KEY(número de tecla de función) GOSUB
número de línea

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de línea es el primer número de línea de una rutina de proceso de la interrupción.
- Si KEY(n) ON está activada, al pulsar la tecla de función n, la sentencia ON KEY(n) GOSUB detiene la ejecución del programa y pasa el control al número de línea especificado.
- La sentencia RETURN vuelve desde la rutina de proceso de la interrupción al punto interrumpido, o a una línea de recomienzo específica, si se ha indicado en la sentencia RETURN.

(Ejemplos)

Ver la Sección 3.6.3.

3.7. Funciones de interrupción del reloj

3.7.1. ON TIME GOSUB (en hora ir a subrutina)

(Función)

La sentencia ON TIME GOSUB define una rutina de interrupción del reloj.

(Formato)

ON TIME GOSUB número de línea

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de línea es el número de línea de comienzo de una rutina de proceso de la interrupción.
- La sentencia RETURN devuelve el control desde la rutina de proceso de la interrupción.

(Ejemplos)

```
1 WIDTH 40,25
5 CLS
10 TIME$="07:29:50"
20 ON TIME GOSUB 200
30 TIME "07:30:00"
40 TIME ON
50 LOCATE 15,10:PRINT TIME$:GOTO 40
60 END
200 BEEP 1
205 LOCATE 15,10:COLOR 2:PRINT TIME$
210 COLOR 7:LOCATE 10,12:INPUT "HORA DE DESPERTAR
!",A$
220 BEEP 0
230 RETURN 60
```

Este programa establece el reloj a 7:29:50 mediante TIME\$, y genera una interrupción a las 7:30 de la mañana. Visualiza el valor del reloj hasta la hora establecida. Al llegar a la hora establecida, llama a la rutina de proceso de la interrupción para visualizar un mensaje de despertar y hacer sonar el timbre electrónico. Pulsando la tecla RETURN se para el timbre y termina el proceso.

3.7.2. TIME (hora)

(Función)

La sentencia TIME establece la hora de interrupción del temporizador.

(Formato)

TIME "hh:mm:ss"

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se especifica la hora de interrupción del temporizador en el formato hh: mm: n.
- Sólo se puede establecer una hora de interrupción del reloj a la vez.

(Ejemplos)

Ver la Sección 3.7.1.

3.7.3. TIME ON/OFF/STOP (hora/activar/desactivar/aplazar)

(Función)

La sentencia TIME ON/OFF/STOP habilita, inhibe o aplaza las interrupciones del reloj.

(Formato)

TIME { ON OFF STOP }

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- TIME ON permite la interrupción del tiempo establecido en la sentencia TIME. Cuando se llega a la hora establecida después de la ejecución de la sentencia TIME ON, se detiene la ejecución del programa y el control se transfiere a la rutina de proceso de la interrupción definida en la sentencia ON TIME GOSUB. La sentencia RETURN vuelve al punto interrumpido, o a una línea de recomienzo específica si se especifica una en la sentencia RETURN.
- TIME OFF inhibe las interrupciones del reloj.
- TIME STOP aplaza temporalmente el proceso de la interrupción. Es decir, después de ejecutarse la sentencia TIME STOP, se acepta una interrupción al tiempo establecido, pero no se llama a la rutina de proceso de la interrupción hasta que no se ejecute la siguiente sentencia TIME ON.

(Ejemplos)

Ver la Sección 3.7.1.

3.7.4. ON INTERVAL GOSUB (en intervalo ir a subrutina)

(Función)

La sentencia ON INTERVAL GOSUB define una rutina de interrupción del reloj por intervalos.

(Formato)

ON INTERVAL GOSUB número de línea

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de línea es el número de línea de comienzo de una rutina de proceso de la interrupción.
- La sentencia RETURN vuelve desde la rutina de proceso de la interrupción al programa que hace la llamada.

(Ejemplos)

```
10 ON INTERVAL GOSUB 50
15 INTERVAL 2
20 TIME$="10:00:00"
30 INTERVAL ON
40 GOTO 40
50 BEEP
60 I=I+1
70 PRINT TIME$,I
80 RETURN
```

Este programa establece el intervalo a 2. Después de transcurrir 2 segundos, se llama a la rutina de interrupción para que imprima el valor del reloj, y entonces el control vuelve al programa que hace la llamada.

RUN	
10:00:02	1
10:00:04	2
10:00:06	3
10:00:08	4
10:00:10	5
10:00:12	6
10:00:14	7
10:00:16	8
10:00:18	9
10:00:20	10

3.7.5. INTERVAL (intervalo)

(Función)

La sentencia INTERVAL fija el intervalo de las interrupciones del reloj de intervalos.

(Formato)

INTERVAL intervalo de interrupción

(Versiones)



(Descripción)

- El intervalo de interrupción puede fijarse en segundos hasta un máximo de 65535.
- Cuando se ejecuta la sentencia INTERVAL en el estado de permiso de interrupciones (INTERRUPT-ENABLED), una interrupción se produce a intervalos del número especificado de segundos y el control se transfiere a la rutina de proceso de interrupciones definida por la sentencia ON INTERVAL GOSUB.
- Se empieza a contar el tiempo cuando se ejecuta la sentencia INTERVAL o cuando se produce una interrupción, y la próxima interrupción se produce cuando concluye el intervalo especificado.
- Si no se ha especificado el intervalo mediante un número entero, el número es transformado al número entero más próximo truncando la parte fraccionaria.

(Ejemplos)

Ver Sección 3.7.4.

3.7.6. INTERVAL ON/OFF/STOP (intervalo activar/ desactivar/aplazar)

(Función)

La sentencia INTERVAL ON/OFF/STOP habilita, inhibe o aplaza las interrupciones del reloj por intervalos.

(Formato)

INTERVAL { ON | OFF | STOP }

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- INTERVAL ON habilita el proceso de interrupciones a intervalos especificados en la sentencia INTERVAL.
- INTERVAL OFF inhibe las interrupciones del temporizador por intervalos.
- INTERVAL STOP aplaza temporalmente el proceso de interrupciones. Es decir, se acepta una interrupción en los intervalos especificados después de ejecutarse la sentencia INTERVAL STOP, pero se suprime la operación de interrupción actual. Se llamará a la rutina de proceso de la interrupción cuando se ejecute la próxima sentencia INTERVAL ON.
- La sentencia INTERVAL OFF debe ejecutarse al final de un programa.

(Ejemplos)

Ver la Sección 3.7.4.

3.8. Funciones de control de línea

3.8.1. ON COM(n) GOSUB (en comunicación ir a subrutina)

(Función)

La sentencia ON COM(n) GOSUB define una rutina de proceso de la interrupción llamada al producirse una interrupción de entrada/salida desde un canal de comunicaciones.

(Formato)

ON COM(número de puerto) GOSUB número de línea

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de línea indica la línea de comienzo de una rutina de proceso de la interrupción llamada cuando se produce una entrada en el puerto de comunicación especificado por el número de puerto.
- El número de puerto debe estar comprendido entre 0 y 4.
- La sentencia RETURN devuelve el control de la rutina de proceso de la interrupción.
- En las funciones de control de línea, COM(n) se define del modo siguiente:

COM0 indica el puerto RS-232C en la unidad principal del FM-7.

COM1 a 4 indica los puertos RS-232C en las unidades de ampliación.

(Ejemplos)

```
10 OPEN"I",#1,"COM0:(S8N2)"
20 ON COM(0) GOSUB 100
30 COM (0) ON
40 GOTO 40
50 END
100 INPUT#1,A$
110 PRINT A$
120 IF A$<>"END" THEN RETURN
130 CLOSE#1
140 RETURN 50
```


3.8.2. COM(n) ON/OFF/STOP (comunicación(n) activar/desactivar/aplazar)

(Función)

La sentencia COM(n) ON/OFF/STOP habilita, inhibe o aplaza las interrupciones de entrada desde un canal de comunicaciones.

(Formato)

COM(número de puerto) {ON | OFF | STOP}

(Versiones)



(Descripción)

- El número de puerto debe estar comprendido entre 0 y 4.
- ON habilita las interrupciones de entrada desde el puerto de comunicación especificado por el número de puerto. Una vez ejecutada esta sentencia, cuando se produce una interrupción en el puerto especificado se invoca la rutina especificada en la sentencia ON COM(n) GOSUB.
- OFF inhibe las interrupciones de entrada desde el puerto de comunicación especificado.
- STOP aplaza temporalmente una interrupción de entrada desde el puerto de comunicación especificado. Se llama a la rutina definida en la sentencia ON COM(n) GOSUB cuando se ejecuta la próxima sentencia COM(n) ON.

(Ejemplos)

Ver la Sección 3.8.1.

3.8.3. OPEN (abrir)

(Función)

La sentencia OPEN abre un fichero para permitir entradas o salidas a través de un canal de comunicación.

(Formato)

OPEN "modo",[#]número de fichero,"COMn:
[(opción)]"

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El modo ha de ser o I (entrada) u O (salida).
- El número de fichero se asigna a un puerto de comunicación RS-232C. Debe estar comprendido entre 1 y 16.
- El nombre del dispositivo ha de estar comprendido entre COM0 a COM4:.
- La opción en paréntesis al dar valores en el orden de c, b, p y s.

c Especifica la velocidad del reloj.

S: Reloj lento (1/64)

F: Reloj rápido (1/16)

b Especifica la longitud de los datos en bits.

8: 8 bits/carácter

7: 7 bits/carácter

p Especifica si hay que hacer una prueba de paridad, y si es impar o par.

N: No hay prueba

O: Prueba de paridad impar

E: Prueba de paridad par

s Especifica el número de bits de stop.

2: 2 bits de stop

1: 1 bit de stop

· Si se omite la opción, se asume (S8N2).

(Ejemplos)

10 OPEN "O",#1,"COM0:(S8N2)"

3.8.4. CLOSE (cerrar)

(Función)

La sentencia CLOSE cierra los ficheros asignados a los canales de comunicación.

(Formato)

CLOSE[#]número de fichero
[, [#]número de fichero]...

(Versiones)



(Descripción)

- El número de fichero es el especificado mediante la sentencia OPEN.

(Ejemplos)

Ver la Sección 3.8.1.

3.8.5. INPUT# (entrada#)

(Función)

La sentencia INPUT# introduce datos desde un canal de comunicación.

(Formato)

INPUT# número de fichero, nombre de variable[, nombre de variable]...

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia introduce datos desde el puerto de comunicación RS-232C especificado por el número de fichero, y asigna los datos a las variables especificadas. Los datos están separados por códigos CR, comas (,) o dos puntos (:), y estos códigos de caracteres no se cargan en las variables.
- Cuando se reciben datos desde una línea de comunicación, se produce una interrupción y los datos se almacenan en un buffer de entrada de 127 bytes. Si el buffer de entrada todavía contiene 127 bytes de datos viejos que no se han leído con una sentencia INPUT#, LINE INPUT o INPUT\$, no se pueden recibir datos adicionales porque el buffer está lleno. Entonces se produce un error de "Buffer Overflow" (Capacidad del buffer superada).
- La sentencia INPUT# recoge datos de la longitud requerida desde el buffer de entrada, y asigna los datos a las variables especificadas. Si el buffer de entrada no contiene datos suficientes, el proceso entra en un bucle hasta que se almacenan más datos en el buffer. Para evitar esto, se le indica al programa que compruebe la condición del buffer mediante la función LOF o EOF antes de ejecutar la sentencia INPUT#.

(Ejemplos)

Ver la Sección 3.8.1.

3.8.6. LINE INPUT# (línea de entrada#)

(Función)

La sentencia LINE INPUT# introduce una línea de datos desde un canal de comunicación.

(Formato)

LINE INPUT# número de fichero, nombre de variable cadena

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia introduce datos desde un puerto de comunicación RS-232C especificado por el número de fichero, y asigna los datos a una variable cadena determinada. La entrada termina cuando aparece un código CR o cuando se han leído 255 caracteres. El código CR no se carga en la variable de cadena.

(Ejemplos)

```
10 OPEN "O", #1, "COM0:(S8N2)"
20 OPEN "I", #2, "COM0:(S8N2)"
30 LINE INPUT "DATA", A$
40 PRINT #1, A$
50 LINE INPUT #2, B$
60 PRINT B$
70 IF B$ = "END" THEN 90
80 GOTO 30
90 CLOSE #1, #2
100 END
```


3.8.7. PRINT# (impresión canal#)

(Función)

La sentencia PRINT# envía datos a un canal de comunicación.

(Formato)

PRINT# número de fichero[,expresión[{ ; | , } expresión]...

O

PRINT# número de fichero,USING
cadena de formato;expresión[{ ; | , } expresión]...

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia saca datos al puerto de comunicación RS-232C especificado por el número de fichero.

(Ejemplos)

```
10 OPEN"O", #1,"COM0:(S8N2)"
20 LINE INPUT "DATA";A$
30 IF A$="END" THEN 60
40 PRINT#1,A$
50 GOTO 20
60 CLOSE#1
70 END
```


3.8.8. LIST (listar)

(Función)

La sentencia LIST saca un listado de programa a un canal de comunicación.

(Formato)

LIST "COMn:[(opción)]"[,[número de línea]
[{|-| ,} [número de línea]]]

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Esta sentencia saca un listado de programa al puerto de comunicación RS-232C especificado por COMn:. El nombre del dispositivo debe estar entre COM0: y COM4:. Ver la descripción de OPEN para las opciones.

(Ejemplos)

LIST"COM0:(S8N2)",20-500

3.9. Funciones aritméticas

3.9.1. ABS (absoluto)

(Función)

La sentencia ABS devuelve el valor absoluto del argumento.

(Formato)

ABS(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica.

(Ejemplos)

```
10 A=-3:B=6
20 PRINT ABS(A)
30 PRINT ABS(B)
40 END
```

Ready

RUN

3

6

Ready

3.9.2. ATN (arcotangente)

(Función)

La sentencia ATN devuelve el arcotangente del argumento.

(Formato)

ATN(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica. El arcotangente se expresa en radianes ($\pi/180$ grados)
- El resultado es un valor entre $-\pi/2$ y $\pi/2$.
- ATN se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=ATN(3.14159/180*A)
30 PRINT B
```

```
Ready
RUN
? 350
1.40853
```

```
Ready
```


3.9.3. COS (coseno)

(Función)

La sentencia COS devuelve el coseno del argumento.

(Formato)

COS(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica, y el argumento debe expresarse en radianes ($\pi/180$ grados).
- COS se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=COS(3.14159/180*A)
30 PRINT B
```

```
Ready
RUN
? 60
.5
```


3.9.4. EXP (exponencial)

(Función)

La sentencia EXP devuelve la constante e elevada a la potencia del argumento.

(Formato)

EXP(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica, y el argumento debe ser menor que 88.0297.
- Si el resultado del cálculo supera el número permitido de dígitos, se produce un mensaje de error "Overflow" (capacidad superada).
- EXP se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=EXP(A)
30 PRINT B
```

```
Ready
RUN
? 5
148.413
```

```
Ready
```


3.9.5. FIX (truncar a entero)

(Función)

La sentencia FIX devuelve la parte entera del argumento.

(Formato)

FIX(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica.
- $\text{FIX}(X)$ da el mismo resultado que $\text{SGN}(X) * \text{INT}(\text{ABS}(X))$.
- $\text{FIX}(X)$ e $\text{INT}(X)$ devuelven el mismo valor cuando X es positivo, pero devuelven distintos valores cuando X es negativo.

(Ejemplos)

```
10 INPUT A
20 B=FIX(A)
30 C=INT(A)
40 PRINT B,C
```

```
Ready
RUN
? -65.4
```

```
-65          -66
```

```
Ready
```


3.9.6. INT (parte entera)

(Función)

La sentencia INT devuelve el mayor número entero menor o igual que el argumento.

(Formato)

INT(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica.
- Hay que tener cuidado con las funciones INT y FIX, ya que no siempre producen el mismo resultado.

(Ejemplos)

```
10 INPUT A
20 B=INT(A)
30 C=FIX
40 PRINT B,C
```

```
Ready
RUN
? -36.8      -36
-37
```

Ready

3.9.7. LOG (logaritmo)

(Función)

La sentencia LOG devuelve el logaritmo natural del argumento.

(Formato)

LOG(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica, y su valor positivo.
- LOG se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=LOG(A)
30 PRINT B
```

```
Ready
RUN
? 59
4.07754
```

```
Ready
```


3.9.8. RND (aleatorio)

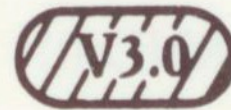
(Función)

La sentencia RND genera un número aleatorio entre cero y uno.

(Formato)

RND[(expresión)]

(Versiones)



(Descripción)

- Se produce la misma secuencia de números aleatorios en cada ejecución de programa; no obstante, la secuencia puede cambiarse mediante el uso de la sentencia RANDOMIZE. (Ver la sentencia RANDOMIZE).
- La expresión debe ser numérica. La secuencia de números aleatorios producidos depende del valor de la expresión como se indica a continuación:

Cuando el argumento es menor que cero, se produce una nueva secuencia de números aleatorios.

Cuando el argumento es cero, se devuelve el último número aleatorio que se había generado previamente.

Cuando se omite el argumento o cuando éste es positivo, se genera el siguiente número aleatorio de la misma secuencia.

(Ejemplos)

10 FOR I=1 TO 3	Ready
20 PRINT RND(1)	RUN
30 NEXT	.591065
40 PRINT	.207991
50 X=RND(-6)	.550967
60 PRINT	
70 FOR I=1 TO 3	
80 PRINT RND(1)	.208935
90 NEXT	.707458
100 PRINT	.353187
110 PRINT RND(0)	
120 END	.353187

Ready

3.9.9. SGN (signo)

(Función)

La sentencia SGN devuelve +1 si el argumento es positivo, 0 si el argumento es cero, y -1 si el argumento es negativo.

(Formato)

SGN(expresión)

(Versiones)



(Descripción)

- La expresión debe ser numérica.

(Ejemplos)

```
10 INPUT A
20 B=SGN(A)
30 PRINT B
Ready
RUN
? -5
-1
```

```
Ready
RUN
? 6
1
```

```
Ready
```


3.9.10. SIN (seno)

(Función)

La sentencia SIN devuelve el seno del argumento.

(Formato)

SIN (expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica, y el argumento debe expresarse en radianes ($\pi/180 \times$ grados).
- SIN se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=SIN(3.14159/180*A)
30 PRINT B
```

```
Ready
RUN
? 60
.866025
```

```
Ready
```


3.9.11. SQR (raíz cuadrada)

(Función)

La sentencia SQR devuelve la raíz cuadrada del argumento.

(Formato)

SQR(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica. No se permiten los valores negativos.
- SQR se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=SQR(A)
30 PRINT B
```

```
Ready
RUN
? 2
1.41421
```

```
Ready
```


3.9.12. TAN (tangente)

(Función)

La sentencia TAN devuelve la tangente del argumento.

(Formato)

TAN(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica, y el argumento debe expresarse en radianes ($\pi/180 \times \text{grados}$).
- TAN se calcula en simple precisión.

(Ejemplos)

```
10 INPUT A
20 B=TAN(3.14159/180*A)
30 PRINT B
```

```
Ready
RUN
? 60
1.73205
```

```
Ready
```


3.9.13. CSNG (convertir a simple)

(Función)

La sentencia CSNG convierte un número de doble precisión a un número equivalente de simple precisión.

(Formato)

CSNG(expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- CSNG convierte el argumento a un número equivalente de 6 dígitos y de simple precisión. Si el valor convertido no se halla dentro de la gama entre $-1.70141E+38$ y $1.70141E+38$, se produce un error de "Overflow" (capacidad superada).

(Ejemplos)

```
10 A#=5649.6517#  
20 B=CSNG(A#)  
30 PRINT A#,B
```

Ready

RUN

5649.6517

5649.65

Ready

3.9.14. CDBL (convertir a doble)

(Función)

La sentencia CDBL convierte un número de simple precisión a un número equivalente de doble precisión.

(Formato)

CDBL(expresión)

(Versiones)



(Descripción)

- CDBL convierte el valor de su argumento a un número equivalente de doble precisión. Sólo se cambia el tipo; la exactitud real del número permanece inalterada.

(Ejemplos)

```
10 A = 4020.12
20 B # = CDBL(A)
30 PRINT A, B #
```

Ready

RUN

4020.12

4020.119873046875

Ready

3.9.15. CINT (convertir a entero)

(Función)

La sentencia CINT convierte un número de simple o de doble precisión a un número entero.

(Formato)

CINT(expresión)

(Versiones)



(Descripción)

- La función CINT redondea la parte decimal del argumento al entero más próximo, considerando a las fracciones menor que 0,5 como 0 y las fracciones igual o mayor que 0,5 como 1.
- Si el valor de la expresión no está comprendido entre -32768 y +32768, se produce un error de "Overflow" (capacidad superada).

(Ejemplos)

```
10 A=35.649:B=-5.528
20 C=CINT(A):D=CINT(B)
30 PRINT C,D
```

Ready

RUN

36

-6

Ready

3.10. Funciones cadena

3.10.1. CHR\$ (carácter\$)

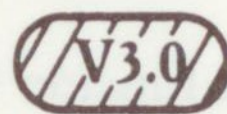
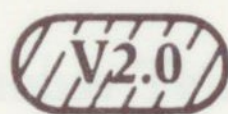
(Función)

La sentencia CHR\$ devuelve un carácter que corresponde al argumento.

(Formato)

CHR\$(expresión)

(Versiones)



(Descripción)

- El valor de la expresión ha de ser un número entre 0 y 255. Ver la tabla de códigos de caracteres para los códigos de caracteres disponibles.
- La función ASC es el contrario de la función CHR\$. La función ASC convierte un carácter a su código de caracteres correspondiente.

(Ejemplos)

```
10 FOR I=&H20TO &H4F
20 PRINT CHR$(I);
30 NEXT
```

Se sacan los caracteres cuyo código esté comprendido entre &H20 a &H4F.

RUN

!"#\$%&'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMNO

3.10.2. HEX\$ (hexadecimal \$)

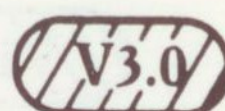
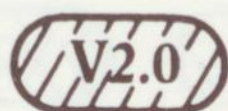
(Función)

La sentencia HEX\$ convierte la parte entera del argumento en su número equivalente en hexadecimal.

(Formato)

HEX\$(expresión)

(Versiones)



(Descripción)

- La expresión debe ser numérica. Si contiene una parte decimal, ésta será redondeada al entero más próximo mediante la ejecución interna de la función INT, antes de la conversión.
- El valor devuelto está en la gama de 0000 a FFFF.

(Ejemplos)

10 INPUT "INTRODUZCA UN N° DECIMAL",A	El número decimal
20 PRINT "&H";HEX\$(A)	introducido se convierte
	en el número hexadecimal
	correspondiente

Ready
RUN
INTRODUZCA UN N°
DECIMAL 186
&HBA

Ready

3.10.3. LEFT\$ (izquierda \$)

(Función)

La sentencia LEFT\$ devuelve los primeros n caracteres del argumento comenzando por la izquierda.

(Formato)

LEFT\$(expresión cadena,número de caracteres)

(Versiones)



(Descripción)

- El número de caracteres debe estar entre 0 y 255. Si es 0, se obtiene una cadena vacía. Si el número especificado de caracteres es mayor que el número de caracteres de la cadena especificada, se obtiene toda la cadena.

(Ejemplos)

```
10 A$="FUJITSU PERSONAL COMPUTER"  
20 B$=LEFT$(A$,7)  
30 PRINT B$
```

```
Ready  
RUN  
FUJITSU
```

```
Ready
```


3.10.4. MID\$ (interior \$)

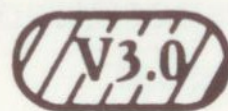
(Formato)

MID\$ (expresión cadena, posición de carácter
[,número de caracteres])

(Función)

La sentencia MID\$, cuando se usa en este formato, devuelve el número especificado de caracteres de la cadena indicada comenzando por el carácter identificado por la posición de carácter.

(Versiones)



(Descripción)

- La posición de carácter se cuenta desde el primer carácter de la cadena especificada, y debe estar entre 1 y 255.
- El número de caracteres ha de estar comprendido entre 0 y 255.
- Cuando el número de caracteres es cero o cuando la posición del carácter excede el número de caracteres de la cadena especificada, se obtiene una cadena vacía. Si se omite el número de caracteres o si es mayor que los caracteres que quedan desde la posición de carácter especificada, se obtiene una cadena formada por todos los caracteres desde tal posición hasta el final.

(Ejemplos)

```
10 A$="FUJITSU PERSONAL COMPUTER"  
20 B$=MID$(A$,9,8)  
30 PRINT B$
```

```
Ready  
RUN  
PERSONAL
```

```
Ready
```


(Formato 2)

MID\$ (variable cadena, posición de carácter
[,número de caracteres])=expresión cadena

(Función)

La sentencia MID\$, cuando se usa en este formato, sustituye la cadena identificada mediante variable cadena por la cadena especificada mediante expresión cadena. La sustitución se inicia en la posición de carácter especificada y continua el número de caracteres especificados. La longitud de la variable cadena permanece igual.

(Versiones)



(Ejemplos)

```
10 A$="FUJITSU PERSONAL COMPUTER"  
20 MID$(A$,9,8)=" FM-7 "  
30 PRINT A$
```

Ready

RUN

FUJITSU FM-7 COMPUTER

Ready

3.10.5. OCT\$ (octal \$)

(Función)

La sentencia OCT\$ convierte la parte entera del argumento en su número equivalente en octal.

(Formato)

OCT\$(expresión)

(Versiones)



(Descripción)

- La expresión debe ser numérica.
- Si la expresión contiene una parte fraccionaria, es redondeada internamente mediante la función INT antes de la conversión.

(Ejemplos)

```
10 INPUT "NUMERO OCTAL", A
20 PRINT OCT$(A)
30 GOTO 10
```

```
Ready
RUN
NUMERO OCTAL50
62
NUMERO OCTAL30
36
NUMERO OCTAL
```

```
Break In 10
Ready
```

El número decimal que se introduce se convierte en su correspondiente número octal que se visualiza.

3.10.6. RIGHT\$ (derecha \$)

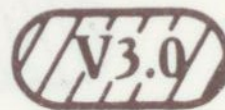
(Función)

La sentencia RIGHT\$ devuelve los primeros n caracteres del argumento comenzando por la derecha.

(Formato)

RIGHT\$(expresión cadena,número de caracteres)

(Versiones)



(Descripción)

- El número de caracteres debe estar entre 0 y 255. Si es 0, se obtiene la cadena vacía. Si el número especificado de caracteres es mayor que el número de caracteres de la cadena especificada, se obtiene toda la cadena.

(Ejemplos)

```
10 A$="FUJITSU PERSONAL COMPUTER"  
20 B$=RIGHT$(A$,8)  
30 PRINT A$  
40 PRINT B$
```

```
Ready  
RUN  
FUJITSU PERSONAL COMPUTER  
COMPUTER
```

```
Ready
```


3.10.7. SPACE\$ (espacio \$)

(Función)

La sentencia SPACE\$ devuelve el número de espacios especificados.

(Formato)

SPACE\$(número de espacios)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de espacios indicado ha de estar comprendido entre 0 y 255. Si se especifica cero, se obtiene la cadena vacía.

(Ejemplos)

```
10 A$=SPACE(5)+"PERSONAL COMPUTER"  
20 PRINT A$
```

Ready

RUN

PERSONAL COMPUTER

Ready

3.10.8. STR\$ (cadena \$)

(Función)

Esta sentencia convierte un argumento numérico en una cadena de caracteres consistente en los caracteres que forman el número.

(Formato)

STR\$ (expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La expresión debe ser numérica.
- La función STR\$ es la inversa de la función VRL.

(Ejemplos)

```
10 A=5649
20 B$="A="+STR$(A)
30 PRINT B$
```

```
Ready
RUN
A=5649
```


3.10.9. STRING\$ (cadena \$)

(Función)

La sentencia STRING\$ devuelve una cadena de caracteres consistente en un carácter repetido el número de veces especificado.

(Formato)

STRING\$ (número de caracteres, { código de carácter expresión de cadena })

(Versiones)



(Descripción)

- El número de caracteres y el código de carácter debe estar entre 0 y 255. Si el número de caracteres es cero, se obtiene una cadena vacía.
- Si la expresión consta de dos o más caracteres, sólo es válido el primero.

(Ejemplos)

```
10 A$=STRING$(20,"A")
```

```
20 B$=STRING$(10,"B")
```

```
30 PRINT A$
```

```
40 PRINT B$
```

```
Ready
```

```
RUN
```

```
AAAAAAAAAAAAAAAAAAAAA
```

```
BBBBBBBBBB
```

```
Ready
```


3.10.10. ASC (ASCII)

(Función)

La sentencia ASC devuelve el código de carácter correspondiente al primer carácter de la cadena especificada.

(Formato)

ASC (expresión de cadena)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El resultado de la función ASC es un número decimal que representa el código del primer carácter de la cadena.
- Si el resultado de la expresión cadena es la cadena vacía, se produce un error "Illegal Function Call" (Llamada ilegal a función).
- La función CHR\$ es la inversa de la función ASC. CHR\$ convierte códigos de caracteres en caracteres.

(Ejemplos)

```
10 A$="A"  
20 A=ASC(A$)  
30 PRINT A
```

```
Ready  
RUN  
65
```


3.10.11. INSTR (en cadena)

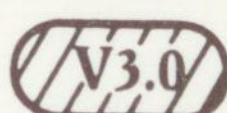
(Función)

La sentencia devuelve la posición del carácter en el que comienzan los caracteres especificados, en la cadena especificada.

(Formato)

INSTR	([posición de comienzo de búsqueda,]expresión cadena 1, expresión de cadena 2)
-------	---

(Versiones)



(Descripción)

- La cadena especificada por la expresión cadena 1 se compara con la expresión cadena 2. Cuando se encuentra la cadena 1 dentro de la cadena 2, la función devuelve la posición del carácter en la que comienza esta coincidencia. Si se omite la posición de comienzo de búsqueda, la búsqueda se realiza desde el primer carácter.
- Si la posición de comienzo de búsqueda es mayor que el número de caracteres de la cadena identificada, o si dicha cadena es una cadena nula, o si no se encuentra la cadena buscada, se obtiene 0.
- Si la expresión cadena 2 es una cadena nula, se obtiene el valor de la posición de comienzo de búsqueda, o en su defecto 1.
- La posición de comienzo de búsqueda debe estar entre 0 y 255. Si no, se produce un error "Illegal Function Call" (Llamada ilegal a función)

(Ejemplos)

```
10 A$="FUJITSU"  
20 A=INSTR(1,A$,"I")  
30 PRINT A$
```

Ready
RUN

4

Ready

3.10.12. LEN (longitud)

(Función)

La sentencia LEN especifica la longitud de la cadena de caracteres especificada.

(Formato)

LEN (expresión cadena)

(Versiones)



(Descripción)

- Si la expresión cadena especificada es igual a la cadena vacía, se devuelve un 0.

(Ejemplos)

```
10 A$="FUJITSU PERSONAL COMPUTER"
```

```
20 A=LEN(A$)
```

```
30 PRINT A
```

Ready

RUN

25

Ready

3.10.13. VAL (valor)

(Función)

La sentencia VAL devuelve una representación numérica de una cadena de caracteres.

(Formato)

VAL (expresión de cadena)

(Versiones)



(Descripción)

- A no ser que la cadena identificada empiece con "+", "-", "&", ".", o un carácter numérico, se devuelve 0. Cuando aparece un carácter que no sea un numeral (0 a F en hexadecimal, o 0 a 7 en octal), se ignoran los caracteres posteriores.
- Se saltan los blancos en la cadena identificada.
- La función STR\$ es el inverso de la función VAL. STR\$ convierte un valor numérico en una cadena de caracteres.

(Ver la Sección 3.10.8.).

(Ejemplos)

```
10 A$="6517"  
20 A=VAL(A$)  
30 B=VAL("&H"+A$)  
40 PRINT A  
50 PRINT B
```

Ready

RUN

6517

25879

Ready

3.11. Funciones Generales

3.11.1. CSRLIN (línea de cursor)

(Función)

La sentencia CSRLIN devuelve la posición vertical en la que se encuentra el cursor.

(Formato)

CSRLIN

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La posición vertical actual del cursor se devuelve como un número de línea.

(Ejemplos)

```
10 LOCATE 20,5
20 A = CSRLIN
30 B = POS(0)
40 LOCATE 5,7
50 PRINT A,B
```

RUN

5

20

Ready

3.11.2. POS (posición)

(Función)

La sentencia POS devuelve la posición horizontal en la que se encuentra el cursor o de la cabeza de impresión.

(Formato)

POS(número de fichero)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Cuando el número de fichero es 0, se devuelve la posición horizontal del cursor.
- Cuando se asigna el número de fichero especificado a la impresora, se devuelve la posición actual de la cabeza de impresión en el buffer de impresión.

(Ejemplos)

Ver los ejemplos de la Sección 3.11.1.

3.11.3. LPOS (posición de línea)

(Función)

La sentencia LPOS devuelve la posición horizontal de la cabeza de impresión.

(Formato)

LPOS (expresión)

(Versiones)

V2.0

V3.0

(Descripción)

La expresión no tiene significado. Se incluye porque el formato lo requiere.

(Ejemplos)

```
10 FOR I=&H20 TO &H3F
20 LPRINT CHR$(I);
30 PRINT LPOS(0);
40 NEXT I
```

Ready
RUN

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22		
23	24	25	26	27	28	29	30	31	32		

Ready

La salida de la impresora es:

!“#\$%&'()*+,-./0123456789:;<=>?

3.11.4. POINT (punto)

(Función)

La sentencia POINT se usa para determinar si un punto se está visualizando o no en una posición especificada.

(Formato)

POINT (posición horizontal, posición vertical)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Si un punto se está visualizando en una posición especificada, se devuelve -1. Si no es así (si el color en esa posición es el del fondo), se devuelve 0.

(Ejemplos)

```
10 PSET(320,100,7)
20 A=POINT(320,100)
30 B=POINT(321,100)
40 PRINT A,B
```


3.11.5. ERR/ERL (número de error/línea de error)

(Función)

La sentencia ERR devuelve un número de error cuando se produce un error, y la sentencia ERL devuelve el número de la línea en la cual se produjo el error.

(Formato)

ERR/ERL

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Cuando se produce un error, la variable del sistema ERR indica el código del error y ERL indica el número de la línea en la cual se produjo el error. Si se produce un error en una sentencia ejecutada en el modo directo, ERL indica 65535 como el número de línea.
- Por lo general, las funciones ERR y ERL se usan para controlar el flujo del programa en una rutina de proceso de error especificada por la sentencia ON ERROR GOTO. Si se ejecuta la sentencia RESUME en la rutina de error, ERR se inicializa a 0; sin embargo, ERL retiene el número de línea del último error hasta que se produce el siguiente.

(Ejemplos)

Ver los ejemplos de la Sección 3.2.27.

3.11.6. VARPTR (puntero de variable)

(Función)

La sentencia VARPTR devuelve la dirección a partir de la cual se almacenan los datos identificados por la variable.

(Formato)

VARPTR(variable)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La variable puede ser numérica, cadena, o matriz, con tal de que se le asigne un valor antes de usar VARPTR. Si no se asigna ningún valor, se produce un error de "Illegal Function Call" (llamada ilegal a función).
- Si se especifica una matriz, los valores deben estar asignados a todas las variables. Esto se debe a que la dirección de matriz cambia cada vez que se asigna un valor nuevo a la variable.
- Los datos se almacenan en la dirección devuelta del siguiente modo:

(1) Datos enteros

0	8 bits más significativos
1	8 bits menos significativos

(3) Datos reales de doble precisión

0	Exponente
1	Mantisa
7	

(2) Datos reales de simple precisión

0	Exponente
1	Mantisa
2	
3	

(4) Datos de caracteres

0	Longitud de la cadena
1	Dirección de la cadena
2	

(Ejemplos)

```
10 A$ = "FUJITSU"  
20 B = VARPTR(A$)  
30 PRINT HEX$(B)
```

Ready
RUN
CCO

Ready

3.11.7. USR (usuario)

(Función)

La sentencia USR llama a una rutina del usuario en lenguaje máquina con la información de entrada proporcionada por el argumento especificado.

(Formato)

USR[numeral](argumento)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El numeral debe ser un entero entre 0 y 9 que haya sido definido con una sentencia DEF o URS.
- Si se omite el numeral, se asume USRO.

(Ejemplo)

Ver la Sección 3.2.2.

3.11.8. PEEK (mirar)

(Función)

La sentencia PEEK devuelve, en decimal, el contenido de la dirección de memoria especificada.

(Formato)

PEEK (expresión)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El valor utilizado en la expresión para especificar una dirección en la memoria principal, ha de estar comprendido entre 0 y 65535.
- PEEK realiza la función inversa de POKE.

(Ejemplos)

```
10 POKE &H5000,&H39
20 B=PEEK(&H5000)
30 PRINT B
```

```
Ready
RUN
57
```

```
Ready
```


3.11.9. FRE(libre)

(Función)

La sentencia FRE devuelve el número de bytes no usados por el BASIC cuando se especifica un entero, o el número de bytes libres del área de cadenas cuando se especifica una cadena de caracteres.

(Formato)

FRE(entero)
FRE(cadena de caracteres)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Cuando se especifica una cadena de caracteres, se vuelve a organizar el área de cadenas antes de que se determine el número de bytes no usados. En esa reorganización se borran todas las cadenas innecesarias, quedando sólo las necesarias.

(Ejemplos)

```
PRINT FRE(0)  
25559
```

```
Ready  
PRINT FRE("A")  
300
```

```
Ready
```


3.11.10. TIME\$ (hora \$)

(Función)

La sentencia TIME\$ devuelve la hora actual del reloj incorporado.

(Formato)

TIME\$

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

Los datos se devuelven en el formato de hh:mm:ss, donde hh es un valor entre 00 y 23 que indica horas, mm es un valor entre 00 y 59 que indica minutos y ss es un valor entre 00 y 59 que indica segundos. El reloj se borra y se pone a 00:00:00 cada vez que se enciende el equipo.

- El formato para establecer la sentencia TIME\$ es hh:mm:ss.

(Ejemplos)

```
10 WIDTH 40,25
20 INPUT "HH:MM:SS";T$
30 TIME$ = T$
40 LOCATE 0,10
50 PRINT TIME$
60 GOTO 40
```

HH:MM:SS? "11:00:00"

11:00:07

Break in 50
Ready

(Formato 2)

TIME

(Versiones)

(Descripción)

- La sentencia TIME se usa como una variable del sistema que indica el tiempo transcurrido en segundos. La variable TIME se inicializa a cero cuando se enciende el equipo, pero puede ser inicializada a otro valor si es necesario.

(Ejemplos)

```
10 WIDTH 40,25
20 LOCATE 0,5
30 PRINT TIME
40 GO TO 20
```

39640

Break in 30
Ready

3.11.11. DATE\$ (fecha \$)

(Función)

La sentencia DATE\$ devuelve la fecha actual del reloj incorporado.

(Formato)

DATE\$

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La fecha se devuelve en el formato de YY/MM/DD, en donde YY, MM y DD son números de dos dígitos que indican el año, el mes y el día respectivamente.
- Se usa el formato YY/MM/DD para establecer la fecha. Cuando se enciende el equipo, DATE\$ se inicializa a un valor establecido en ROM. Para usarla de una forma efectiva, debe volver a ser inicializada. YY representa los últimos dos dígitos del año (por ejemplo, 81 representa 1981).

(Ejemplos)

```
10 WIDTH 40,25
20 DATE$ = "81/10/05"
30 TIME$ = "23:59:57"
40 LOCATE 0,5:PRINT DATE$,TIME$
50 GOTO 40
```

81/10/05 23:59:59

Break in 40
Ready

(Formato 2)

DATE

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- La sentencia DATE se usa como una variable del sistema que indica cuantos días han transcurrido desde el principio del año. (25 enero = día 25).

(Ejemplos)

```
10 DATE$="81/10/07"  
20 PRINT DATE
```

Ready

RUN

280

Ready

3.11.12. SPC (espacio)

(Función)

La sentencia SPC imprime el número de espacios especificado.

(Formato)

SPC(número de espacios)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de espacios debe estar entre 0 y 255.
- La función SPC sólo puede usarse en las sentencias PRINT, LPRINT y PRINT#.

(Ejemplos)

```
10 PRINT "FUJITSU"SPC(3)"PERSONAL"SPC(4)  
"COMPUTER"
```

Ready

RUN

FUJITSU

PERSONAL

COMPUTER

Ready

3.11.13. TAB (tabulador)

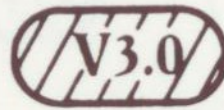
(Función)

La sentencia TAB imprime espacios hasta la posición de columna especificada.

(Formato)

TAB(posición de columna)

(Versiones)



(Descripción)

- La posición de columna debe estar entre 0 y 32767. Si la posición de columna especificada excede el número de columnas por línea, el valor especificado se divide por el número de columnas por línea y se usa el resto. Si la posición de columna especificada está a la izquierda de la posición actual, se imprimen espacios hasta la posición especificada de la línea siguiente.
- La función TAB sólo puede usarse en las sentencias PRINT, LPRINT y PRINT #.

(Ejemplos)

```
10 INPUT "NOMBRE,TELEFONO";A$,B$  
20 PRINT "NOMBRE";TAB(20);"TELEFONO"  
30 PRINT A$;TAB(20);B$
```


3.11.14. SCREEN (pantalla)

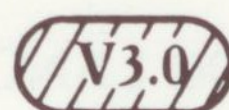
(Función)

La sentencia SCREEN devuelve el código de carácter o el atributo del carácter en la posición especificada.

(Formato)

SCREEN(posición horizontal,posición vertical
[,indicador de selección])

(Versiones)



(Descripción)

- Las posiciones horizontal y vertical identifican la posición del carácter deseado en pantalla.
- El indicador de selección debe ser 0 o 1. Si está puesto a 0 o si se omite, se devuelve el código de carácter del carácter identificado. Si está en 1, se devuelve el atributo de la posición del carácter. El atributo se interpreta del siguiente modo: Dividir el atributo por 8. Un cociente de 0 identifica una visualización normal; un cociente de 1 identifica video inverso. El resto es el código de paleta. Si no se visualiza ningún carácter en la posición especificada, se devuelve 0, independientemente de si el conmutador de selección está a 0 o 1.

(Ejemplos)

```
10 CLS
20 LOCATE 0,0
30 PRINT "FUJITSU PERSONAL COMPUTER"
40 LOCATE 0,5
50 FOR I=24 TO 0 STEP-1
60 A=SCREEN(I,0)
70 PRINT CHR$(A);
80 NEXT
```

FUJITSU PERSONAL COMPUTER

RETUPMOC LANOSREP USTIJUF

Ready

3.12. Funciones de entrada/salida

3.12.1. CVI/CSV/CVD (CVI: convertir a entero, CSV: convertir a simple, CVD: convertir a doble) (Disco).

(Función)

La sentencia CVI/CSV/CVD convierte la cadena de caracteres especificada a sus datos numéricos equivalentes.

(Formato)

CVI (cadena de caracteres de 2 bytes) CSV (cadena de caracteres de 4 bytes) CVD(cadena de caracteres de 8 bytes)
--

(Versiones)



(Descripción)

- Puesto que todos los datos leídos del fichero de disco directo son del tipo de cadena, si se van a manejar como numéricos, antes deben ser convertidos. CVI, CSV y CVD se usan para convertir los datos de cadena en datos numéricos.
- CVI convierte una cadena de caracteres de 2 bytes en un entero; CSV convierte una cadena de caracteres de 4 bytes en un número de simple precisión; CVD convierte una cadena de caracteres de 8 bytes en un número de doble precisión.

(Ejemplos)

```
10 OPEN "R",#1,"TESTDATA"  
20 FIELD #1,2 AS I$,4 AS S$,8 AS D$  
30 INPUT "ENTRE DATOS";I%,S,D#  
40 LSET I$=MKI$(I%)  
50 LSET S$=MKS$(S)  
60 LSET D$=MKD$(D#)  
70 PUT #1,1  
80 PRINT  
90 GET #1,1  
100 I% = CVI(I$)  
110 S = CVS(S$)  
120 D# = CVD(D$)  
130 PRINT I$,S,D#  
140 CLOSE #1  
150 END
```

RUN

ENTRE DATOS 2,32.12,63.25874568

2

32.12

63.25874568

3.12.2. MKI\$/MKS\$/MKD\$ (MKI\$: hacer entero \$, MKS\$: hacer simple \$, MKD\$: hacer doble \$) (Disco).

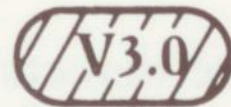
(Función)

La sentencia MKI\$/MKS\$/MKD\$ convierte datos numéricos a su cadena de caracteres equivalente.

(Formato)

MKI\$(entero) MKS\$(número de simple precisión) MKD\$(número de doble precisión)
--

(Versiones)



(Descripción)

- Todos los datos numéricos colocados en el buffer de ficheros de acceso directo deben convertirse en datos cadena equivalentes. Estas funciones se usan para hacer las conversiones necesarias.
- Para convertir datos numéricos en datos cadena, su representación interna se trata como una secuencia de códigos de caracteres.
- Los datos numéricos se convierten a datos de cadena al tratar la representación interna de los datos numéricos como una secuencia de códigos de caracteres.
- MKI\$ convierte un entero en una cadena de 2 bytes; MKS\$ convierte un número de simple precisión en una cadena de 4 bytes; MKD\$ convierte un número de doble precisión en una cadena de 8 bytes.

(Ejemplos)

Ver la Sección 3.12.1.

3.12.3. EOF (fin de fichero)

(Función)

La sentencia EOF devuelve un valor que indica la condición EOF (fin de fichero) o la condición no EOF.

(Formato)

EOF(número de fichero)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Cuando se detecta el fin (EOF) del fichero de entrada identificado, se devuelve -1 (cierto). De lo contrario, se devuelve 0 (falso).
- El fichero indicado por el número de fichero debe ser un fichero de disco, fichero de cinta cassette o un fichero secuencial RS-232C.
- Cuando el número de fichero especificado se asigna a un puerto de comunicaciones RS-232C, se devuelve 0 cuando el buffer está lleno y se devuelve -1 cuando el buffer no está lleno.
- La función EOF no se puede utilizar con el fichero de teclado (cuyo número se asigna por KYBD:).

(Ejemplos)

```
10 OPEN "I", #1, "DATA2"  
20 INPUT #1, A$  
30 PRINT A$  
40 IF EOF(1) THEN 50 ELSE 20  
50 CLOSE #1  
60 END
```

Este ejemplo asume la existencia de un fichero secuencial que se llama DATA2.

```
Ready  
RUN  
822145  
822101  
235  
63254  
123485  
125468  
3256  
214  
3256  
7896
```


3.12.4. LOF (longitud de fichero de disco)

(Función)

La sentencia LOF devuelve el número de caracteres almacenado en el buffer de entrada.

(Formato)

LOF(número de fichero)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- Se devuelve el número de caracteres almacenados en el buffer del fichero de entrada identificado.
- La función LOF no se puede usar para el fichero de teclado (cuyo número se asigna por KYBD:).
- Cuando el número de fichero indica un fichero directo, se devuelve el número del registro más alto.

(Ejemplos)

```
10 OPEN "R", #1, "TESTDATA"  
20 FIELD #1,2 AS I$,4 AS S$,8 AS D$  
30 GET #1,1  
40 A = LOF(1)  
50 PRINT A  
60 CLOSE  
70 END
```


3.12.5. LOC (contador de ubicación del fichero de disco) (Disco)

(Función)

La sentencia LOC da el número del registro en el fichero directo especificado al que se va a acceder mediante la siguiente sentencia GET o PUT.

(Formato)

LOC(número de fichero)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El fichero identificado por el número de fichero debe ser un fichero de disco, que se haya abierto en el modo directo de entrada/salida ("R").
- El número de registro devuelto identifica el registro siguiente al utilizado por la última sentencia GET o PUT. Por consiguiente el resultado es el mismo que cuando se ejecuta una sentencia GET o PUT sin especificar el número de registro.

(Ejemplos)

```
10 OPEN "R", #1, "TESTDATA"  
20 FIELD #1,2 AS I$,4 AS S$,8 AS D$  
30 PUT #1  
40 A =LOC(1)  
50 PRINT A  
60 CLOSE #1  
70 END
```

Ready
RUN

2

3.12.6. DSKI\$ (inicializar disco \$) (Disco)

(Función)

La sentencia DSKI\$ pone el contenido del sector especificado en el buffer directo del sistema.

(Formato)

DSKI\$(número de unidad de disco,número de pista,número de sector)

(Versiones)



(Descripción)

- Antes de usar la función DSKI\$, una variable de cadena debe asignarse al buffer aleatorio del sistema mediante una sentencia FIELD.
- Los números de unidad de disco que pueden especificarse son de 0 a 3.
- Para un minidisquete flexible (5"), los números de pista especificables son de 0 a 39; para un disco floppy estándar (8"), los números de pista especificables son de 0 a 76.
- Cuando se usa un minidisquete flexible, los números de sector especificables son de 1 a 32. Los sectores 17 al 32 son equivalentes a los sectores 1 al 16 en la otra cara del disco. Para un disco flexible estándar, los números de sector especificables son de 1 a 52. Los sectores 27 al 52 son equivalentes a los sectores 1 al 26 en la otra cara del disco.
- El buffer directo del sistema es un área de 256 bytes reservada por el sistema.

(Ejemplos)

10 FIELD #0,128 AS A\$,128 AS B\$ Se asigna el contenido del
20 DUMMY\$=DSKI\$(0,10,1) sector 1, pista 10, unidad de
disco 0 a la variable
cadena DUMMY\$.

3.12.7. DSKF (fichero de disco) (Disco)

(Función)

La sentencia DSKF devuelve el número de grupos (clusters) disponibles en el disco especificado.

(Formato)

DSKF(unidad de disco)

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de la unidad de disco debe estar entre 0 y 3.

(Ejemplos)

```
10 A=DSKF(0)
20 PRINT A
30 END
```

```
Ready
RUN
93
```


3.12.8. INPUT\$ (entrada \$)

(Función)

La sentencia INPUT\$ introduce una cadena formada por el número de caracteres especificado, procedente del teclado o de un fichero especificado.

(Formato)

`INPUT$(número de caracteres[, # ]número de fichero))`

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- El número de caracteres debe estar comprendido entre 0 y 255.
- La cadena se introduce desde un fichero cuando se especifica el número de fichero, o desde el teclado cuando no se especifica ningún número de fichero. No se visualizan los caracteres introducidos desde el teclado. Esta es la diferencia entre la sentencia INPUT\$ y la sentencia INPUT.
- Cuando se introducen caracteres desde el teclado o desde un puerto de comunicaciones RS-232C, si el buffer contiene el número de caracteres especificado, éstos son leídos desde el principio del buffer. Si el buffer está vacío o si contiene menos caracteres de los especificados, el BASIC espera hasta que el buffer contenga los caracteres necesarios.
- INPUT\$ puede leer todos los caracteres de control con excepción de la tecla **PREAK** (la tecla **STOP** en las versiones **V1.0** y **V2.0**), así que también puede leer los códigos CR y LF, que no pueden ser leídos por las sentencias INPUT y LINE INPUT sin modificación.

(Ejemplos)

```
10 A$=INPUT$(3)
20 PRINT A$
```

Se introducen tres caracteres desde el teclado, y se sacan los datos de entrada.

```
Ready
RUN
123
```

```
Ready
```


3.12.9. INKEY\$ (tecla de entrada \$)

(Función)

La sentencia INKEY\$ devuelve el carácter correspondiente a la tecla que se pulse a continuación, y un carácter nulo mientras no se pulse una tecla.

(Formato)

INKEY\$

(Versiones)

V1.0

V2.0

V3.0

(Descripción)

- INKEY\$ devuelve una cadena de caracteres nulos si está vacío el buffer del teclado. Si no está vacío, lee un carácter del principio del buffer y lo devuelve. INKEY\$ puede leer todos los caracteres de control con excepción de la tecla BREAK (la tecla STOP en las versiones V1.0 y V2.0), pero no visualiza el carácter leído.

(Ejemplos)

```
10 A$=INKEY$
20 IF A$="" THEN 10
30 PRINT A$;"=";HEX$(ASC(A$))
40 GOTO 10
```

Este programa visualiza el código ASCII de un carácter introducido desde el teclado.

Ready

RUN

1=31

3=33

2=32

5=35

R=52

K=4B

J=4A

G=47

Break In 10

Ready

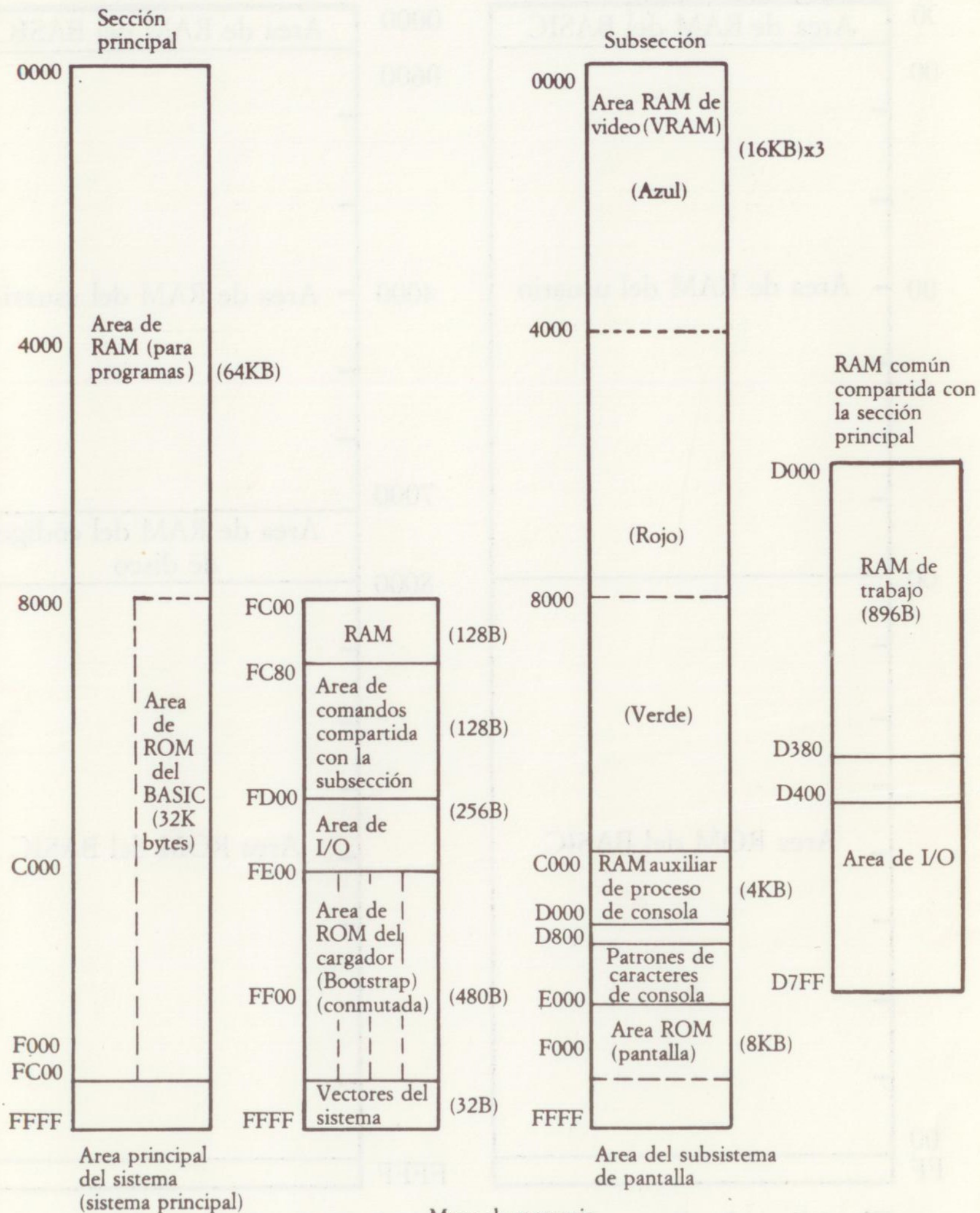
APENDICES

- MAPA DE MEMORIA
- CODIGOS DE CARACTERES
- MENSAJES DE ERROR DEL F-BASIC
- INSTRUCCIONES DEL F-BASIC

APENDICES

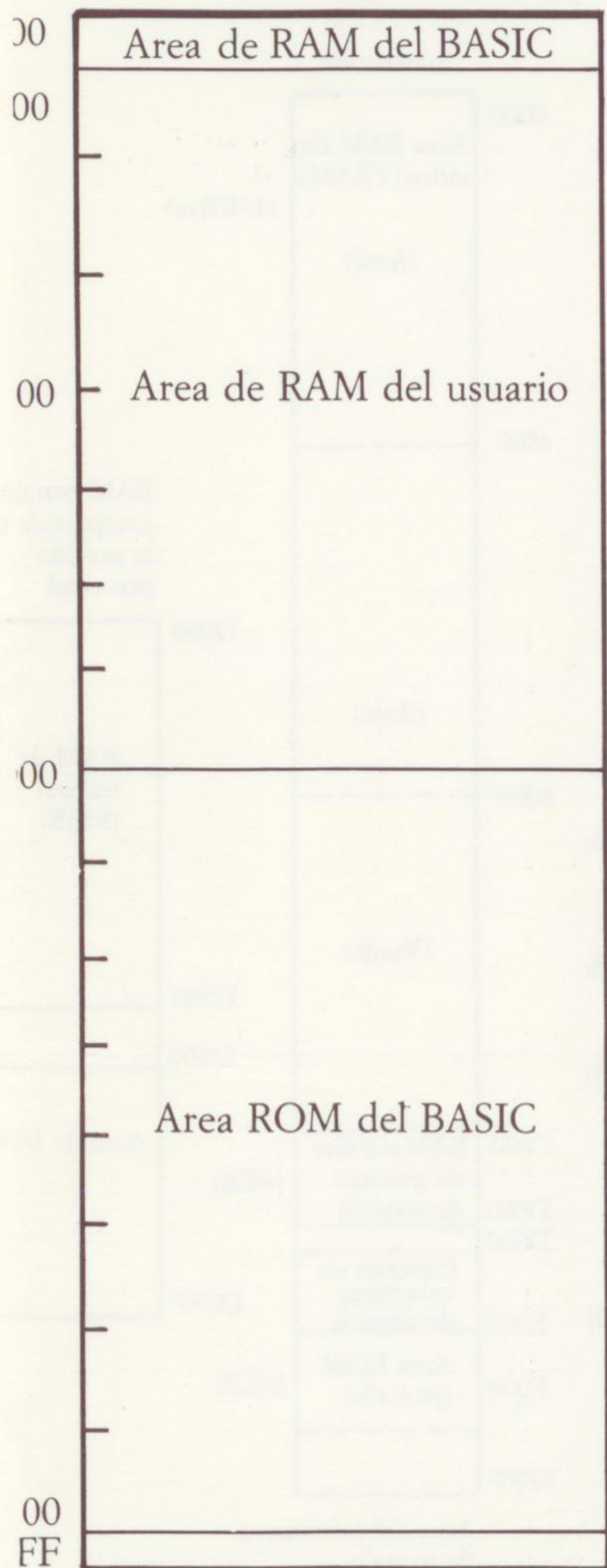
MAPA DE MEMORIA
CODIGOS DE CARACTERES
MENSAJES DE ERROR DEL F-BASIC
INSTRUCCIONES DEL F-BASIC

Apéndice A Mapa de Memoria

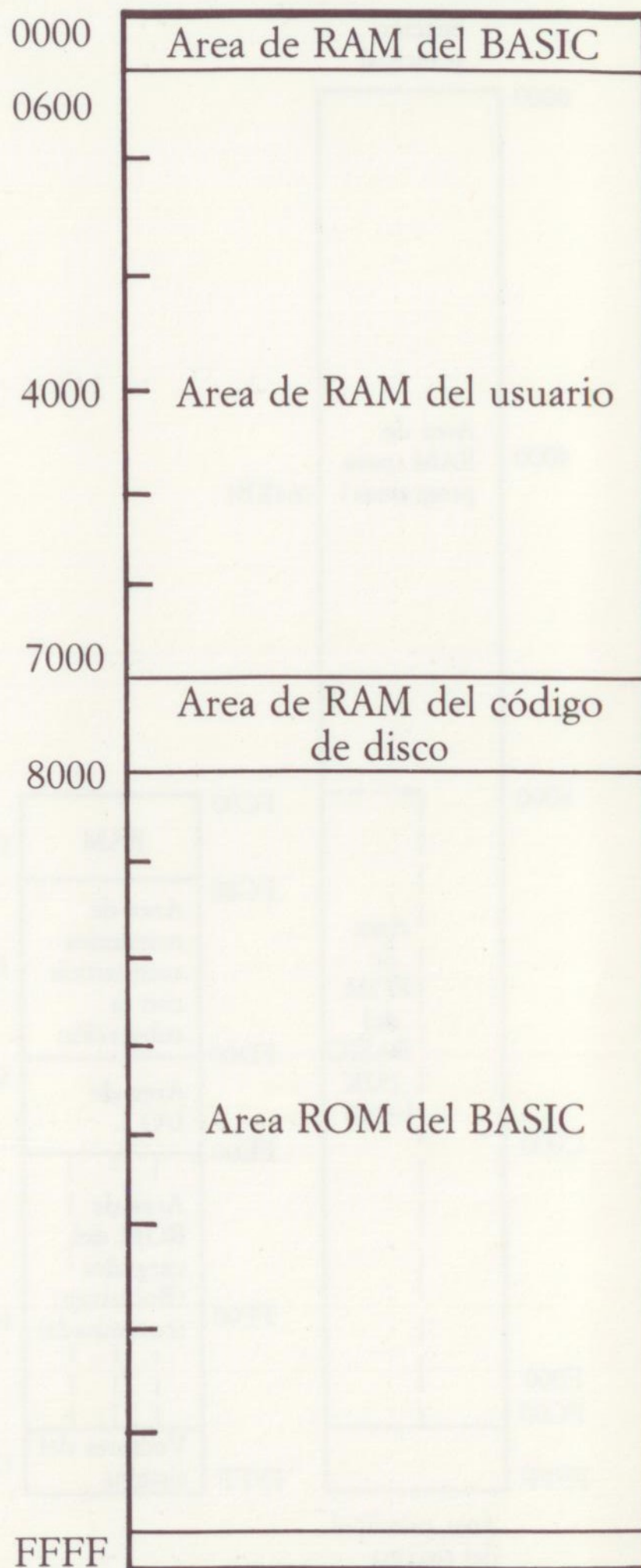


La disposición de memoria de la sección principal es la siguiente cuando se ejecuta F-BASIC V1.0.

Modo ROM

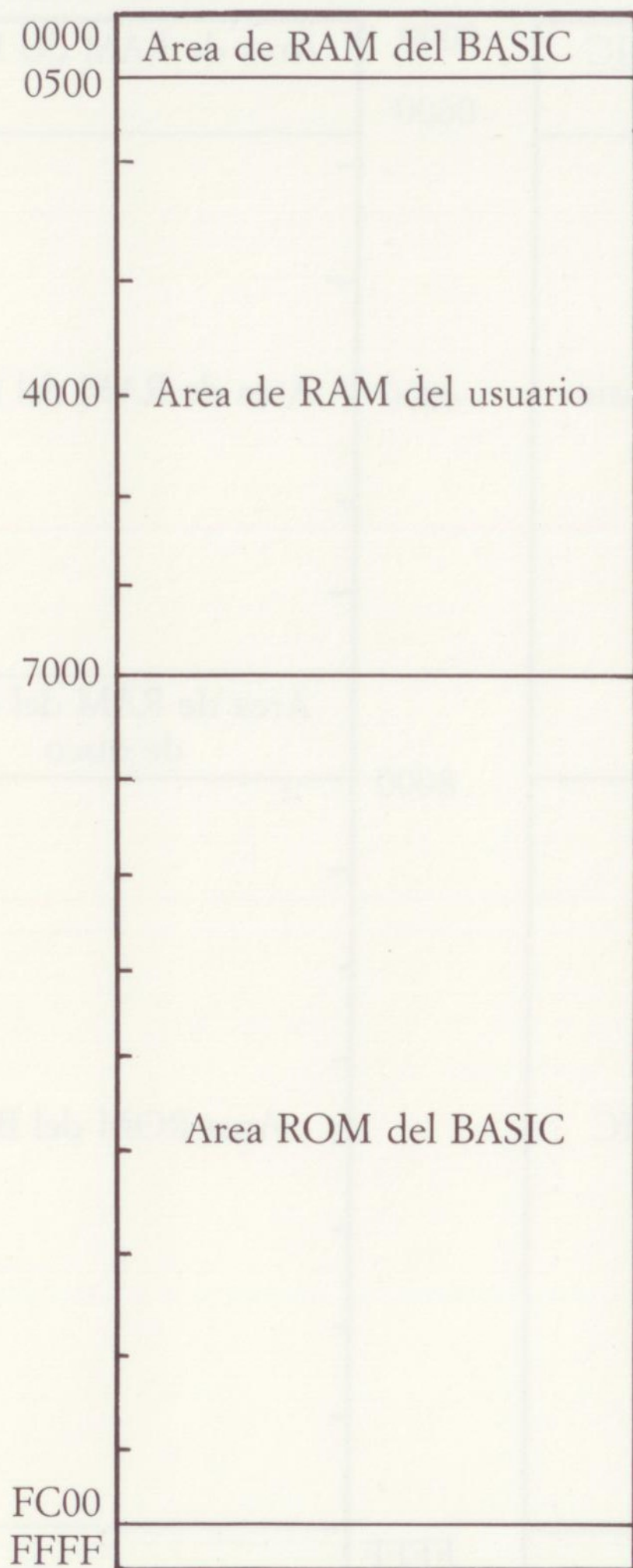


Modo DISCO

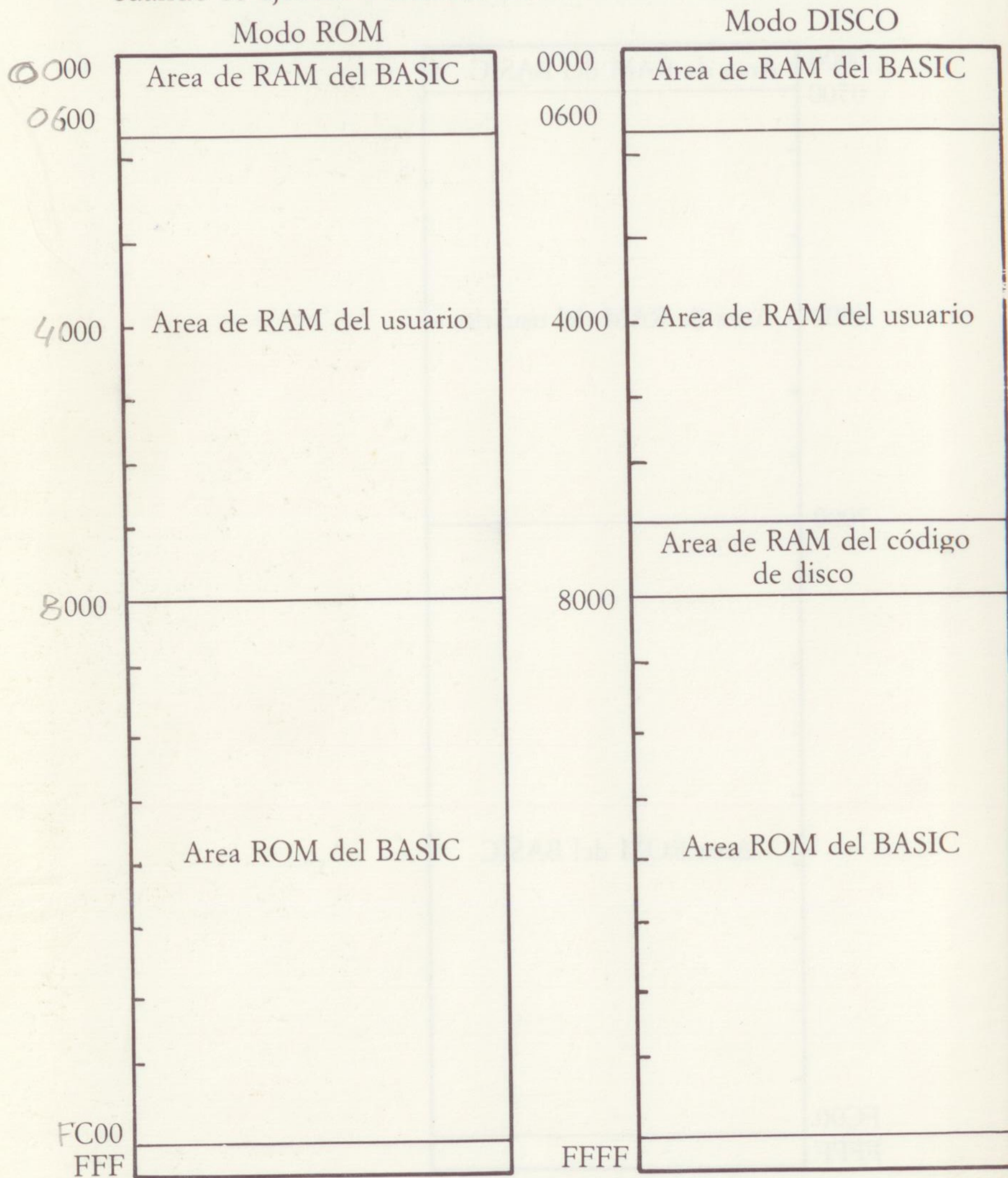


El código de disco se carga en RAM desde el disco del sistema al arrancar el sistema.

La disposición de memoria de la sección principal es la siguiente cuando se ejecuta F-BASIC V2.0.



La disposición de memoria de la sección principal es la siguiente cuando se ejecuta F-BASIC V3.0.



El código de disco se carga en RAM desde el disco del sistema al arrancar el sistema.

Apéndice B Códigos de Caracteres

ORDEN SUP ORDEN INFER	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D _E		0	@	P	\	p		3/4	q	1/4	ℓ	ρ	Φ	o
1	S _H	D ₁	!	1	A	Q	a	q	➤	2/3	ù	§	β	δ	℥	1
2	S _X	D ₂	"	2	B	R	b	r	⬅	1/3	ç	†	γ	τ	Ω	2
3	E _X	D ₃	#	3	C	S	c	s	┐	É	"	φ	δ	υ	≡	3
4	E _T	D ₄	\$	4	D	T	d	t	—	Tm	l	ì	€	φ	±	4
5	E _Q	N _K	%	5	E	U	e	u	≠	ä	Ä	ñ	ſ	χ	≥	5
6	A _K	S _N	&	6	F	V	f	v	≡	ı	Ö	ò	℥	Ψ	≤	6
7	B _L	E _B	'	7	G	W	g	w	œ	ó	Ü	à	θ	ω	ℓ	7
8	B _S	C _N	(	8	H	X	h	x	œ	ú	Æ	æ	ℓ	ℓ	⊠	8
9	H _T	E _M	)	9	I	Y	i	y	Š	ë	À	β	κ	Δ	÷	9
A	L _F	S _B	*	:	J	Z	j	z	š	ü	Ñ	ä	λ	Ⓜ	≈	(
B	H _M	E _C	+	:	K	[	k	{	đ	à	ø	ö	ſ	^	⊠	)
C	C _L	→	.	<	L	\	l	ı	š	ë	i	ü	γ	≡	‰	+
D	C _R	←	—	=	M	}	m	}	⌘	ı	ı	à	ξ	π	v	—
E	S _O	↑	.	>	N	^	n	~	†	ö	£	è	o	&	√	×
F	S _I	↓	/	?	O	—	o	D _L	↓	ü	1/2	é	℥	↔	∞	/

Apéndice C Mensajes de Error del F-BASIC

Código de Error	Mensaje de Error	Significado
01	Next Without For	Falta la sentencia FOR para una sentencia NEXT.
02	Syntax Error	Se ha introducido de forma incorrecta el comando o la sentencia
03	Return Without GOSUB	La sentencia RETURN ha aparecido antes de que una subrutina fuese llamada por GOSUB.
04	Out of Data	No hay datos para que la sentencia READ los lea.
05	Illegal Function Call	Se ha llamado de forma ilegal a una función o sentencia.
06	Overflow	El valor del entero o el valor del número real supera el límite, o es demasiado grande el valor que se va a asignar. La gama permitida de enteros varía entre -32768 y 32767 y la gama permitida de números reales varía entre -1.7014E+38 y 1.70141E+38.
07	Out of Memory	No hay más espacio disponible en memoria.
08	Undefined Line Number	No está definido el número de línea especificado.
09	Subscript Out Of Range	Un valor de subíndice en una variable de matriz no está dentro de la gama de 0 al límite superior declarado.

Código de Error	Mensaje de Error	Significado
10	Duplicate Definition	Se declara dos veces el mismo nombre de matriz o de función del usuario.
11	Division by Zero	El divisor en una operación de división es cero.
12	Illegal Direct	Se ha usado una sentencia inhibida en el modo directo.
13	Type Mismatch	El tipo de variable o constante no concuerda. (Se especifica una operación que combina valores numéricos y valores de cadena). Este error puede producirse en una sentencia LET, en el argumento de una función o en otros sitios.
14	Out Of String Space	El área de cadenas está lleno.
15	String Too Long	Una constante o variable cadena excede 256 caracteres.
16	String Formula Too Complex	Una expresión cadena es demasiado compleja. La profundidad de anidado de una expresión de cadena se limita a 10.
17	Can't Continue	El programa no puede ejecutar un comando CONT.
18	Undefined User Function	Se ha hecho referencia a una función no definida.
19	No Resume	La rutina de proceso de error no contiene RESUME.

Código de Error	Mensaje de Error	Significado
20	Resume Without Error	Se ha intentado la ejecución de una sentencia RESUME antes de producirse un error.
21	Unprintable Error	Se ha producido un error para el cual no hay preparado ningún mensaje de error.
22	Missing Operand	Falta un operando necesario.
23	For Without Next	Falta la sentencia NEXT para la sentencia FOR.
24	While Without Wend	Falta la sentencia WEND para la sentencia WHILE.
25	Wend Without While	Falta la sentencia WHILE para la sentencia WEND.
26	Bubble Full	El cassette de burbuja está lleno y no puede almacenarse más datos.
50	Bad File Number	Se ha usado el número de un fichero no abierto.
51	Bad File Mode	Se ha intentado usar un fichero en un modo diferente del modo en que fue abierto.
52	File Already Open	Un fichero se ha abierto dos veces.
53	Device I/O Error	Se ha producido un error de E/S en el dispositivo.
54	Input Past End	Se ha ejecutado una sentencia INPUT después de leerse todos los datos del fichero.

Código de Error	Mensaje de Error	Significado
55	Bad File Descriptor	Se ha especificado de forma incorrecta el descriptor de ficheros.
56	Direct Statement In File	Un fichero del programa almacenado en código ASCII contiene una sentencia directa.
57	File Not Open	No se ha abierto todavía el fichero.
58	Bad Data in File	Es ilegal el formato de los datos en el fichero.
59	Device in Use	Se ha intentado abrir un fichero en un dispositivo ya en uso.
60	Device Unavailable	No está preparado el dispositivo de entrada/salida.
61	Buffer Overflow	El buffer de E/S está lleno.
62	Protected Program	Se ha intentado escribir una modificación en un fichero protegido.
63	File Not Found	No se encuentra el nombre del fichero especificado.
64	File Already Exists	Ya existe el nombre del fichero especificado.
65	Directory Full	El área del directorio está llena y no se pueden catalogar nuevos ficheros.
66	Too Many Open Disk Files	Se ha intentado abrir más ficheros que los reservados.
67	Disk Full	El disco está lleno, y no pueden almacenarse más datos.
68	Field Overflow	El campo rebasa 256 bytes de largo.

Código de Error	Mensaje de Error	Significado
69	String Not Fielded	Se ha usado LSET o RSET en una variable que no fue declarada por una sentencia FIELD.
70	Bad Record Number	No se puede encontrar el número de registro especificado.
71	Bad File Structure	Es incorrecta la estructura de fichero.
72	Drive Not Ready	La unidad de disco especificada no está preparada.
73	Disk Write Protected	El disco está protegido contra escritura.

Nota: Los mensajes de error 65 al 73 se refieren al modo DISCO.

APENDICE D

INSTRUCCIONES DEL F-BASIC

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Comando	AUTO	Genera números de línea automáticamente para cada línea.	o	o	o
	DELETE	Borra líneas de un programa	o	o	o
	LIST	Visualiza en pantalla parte o todo un programa que está en memoria.	o	o	o
	LLIST	Imprime por impresora parte o todo un programa que está en memoria.	x	o	o
	UNLIST	Especifica las líneas que no deben listarse.	o	o	o
	RENUM	Renumeración de líneas de programa.	o	o	o
	NEW	Elimina todos los programas de memoria e inicializa todas las variables.	o	o	o
	CLEAR	Inicializa cada variable numérica a 0 y cada variable cadena a una cadena vacía; luego fija el límite superior de memoria utilizado por el BASIC tal como se especifique en el operando(s).	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Comando	CONT	Reanuda la ejecución de un programa suspendido después de pulsar la tecla BREAK (tecla STOP en V1.0 y V2.0) o después de ejecutarse una sentencia STOP o END.	o	o	o
	RUN	Ejecuta un programa en memoria o fichero.	o	o	o
	LOAD	Carga un programa en memoria desde un fichero especificado.	o	o	o
	LOAD?	Verifica las sumas de prueba de un fichero almacenado en cinta de casete.	o	o	o
	SAVE	Guarda un programa de memoria en un fichero.	o	o	o
	FILES	Visualiza el listado del directorio de un dispositivo especificado.	o	o	o
	NAME	Cambia el nombre de un fichero en un disco flexible.	o	o	o
	KILL	Elimina un fichero de un disco flexible o casete burbuja.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Comando	MERGE	Combina un programa en memoria con otro de un fichero especificado.	o	o	o
	SKIPF	Mueve una cinta casete hacia adelante al fichero que sigue a un fichero especificado.	o	o	o
	DSKINI	Inicializa el directorio de un disco flexible.	o	o	o
	EXEC	Ejecuta un programa en lenguaje máquina	o	o	o
	LOADM	Carga un programa en lenguaje máquina a memoria y lo ejecuta.	o	o	o
	SAVEM	Almacena un programa en lenguaje máquina desde memoria en un fichero especificado.	o	o	o
	HARDC	Produce una copia impresa de los datos visualizados.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Comando	MON	Cambia al nivel de comandos por monitor desde el nivel de BASIC. M: Modifica contenido de memoria. G: Bifurca a una dirección especificada. R: Visualiza y modifica contenido de los registros. D: Visualiza el contenido de 64 bytes de datos, empezando en una dirección especificada.	o	o	o
	TERM	Pone el modo de operación del FM-7 en modo terminal.	o	o	o
	EDIT	Visualiza en pantalla una línea específica.	o	o	o
Sentencia General	DEF FN	Define y nombra una función de usuario.	o	o	o
	DEF USER	Especifica la dirección de comienzo de un programa en lenguaje máquina.	o	o	o
	DEF INT	Declara un tipo de variable como entero.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia General	DEF SNG	Declara un tipo de variable como de simple precisión.	o	o	o
	DEF DBL	Declara un tipo de variable como de doble precisión.	o	o	o
	DEF STR	Declara un tipo de variable como de cadena.	o	o	o
	REM	Especifica comentarios en un programa. REM puede ser sustituido por un apóstrofe(').	o	o	o
	END	Termina la ejecución de programa, cierra todos los ficheros abiertos y vuelve al nivel de comandos.	o	o	o
	FOR NEXT	Ejecuta repetidamente una serie de instrucciones.	o	o	o
	GOSUB	Llama a una subrutina y al final de la subrutina devuelve el control a la sentencia que sigue a GOSUB.	o	o	o
	GOTO	Bifurca a un número de línea especificado de manera incondicional.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia General	ON GOTO	Bifurca a uno de varios números de línea especificados según el valor de una expresión.	o	o	o
	RETURN	Termina una subrutina y devuelve el control al programa que llama.	o	o	o
	STOP	Detiene ejecución del programa y devuelve el BASIC al nivel de comandos.	o	o	o
	ON GOSUB	Llama a una subrutina empezando con uno de varios números de línea especificados, según el valor de una expresión.	o	o	o
	IF THEN ELSE	Selecciona una sentencia que se va a ejecutar según el valor de una expresión.	o	o	o
	WHILE WEND	Ejecuta repetidamente una serie de instrucciones, según una condición.	o	o	o
	LET	Asigna el resultado de una expresión a la derecha del signo -, a una variable a la izquierda.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia General	SWAP	Intercambia los valores de dos variables.	o	o	o
	DIM	Especifica el número y tamaño de dimensiones de una variable de matriz para reservar espacio en memoria para la variable.	o	o	o
	POKE	Escribe datos en una dirección de memoria especificada.	o	o	o
	DATA	Almacena constantes numéricas y/o cadena que van a ser leídas por sentencias READ.	o	o	o
	READ	Lee constantes definidas por sentencias DATA en variables.	o	o	o
	RESTORE	Hace que la lectura empiece a partir de la primera sentencia DATA.	o	o	o
	LSET,RSET	Transfiere datos de cadena a un buffer de ficheros aleatorios.	o	o	o
	RANDOMIZE	Altera una secuencia de números aleatorios.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia General	ERROR	Simula una ocurrencia de error en BASIC o permite que el usuario defina los números de error.	o	o	o
	ON ERROR GOTO	Permite tratar los errores	o	o	o
	RESUME	Reanuda la ejecución del programa después de procesar errores.	o	o	o
	BEEP	Hace que el altavoz incorporado emita un tono.	o	o	o
	MOTOR	Controla el motor de una grabadora de casete.	o	o	o
	TRON	Traza la ejecución de un programa.	o	o	o
	TROFF	Detiene el trazado de un programa.	o	o	o
	CHAIN	Transfiere variables de un programa en memoria a otro programa especificado, y luego lo ejecuta.	x	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia General	COMMON	Especifica las variables que se van a pasar al programa enlazado por una sentencia CHAIN.	x	o	o
	ERASE	Borra una variable o variable de matriz de un programa.	x	o	o
Sentencia de entrada /salida	INPUT	Lee datos introducidos por el teclado.	o	o	o
	LINE INPUT	Lee una línea completa de caracteres (255 o menos) en una variable cadena sin delimitar caracteres.	o	o	o
	PRINT	Visualiza en pantalla el resultado de expresiones. (Puede usarse un signo de interrogación (?) en vez de la palabra clave PRINT).	o	o	o
	LPRINT	Envía datos a la impresora.	x	o	o
	PRINT USING	Visualiza en pantalla cadenas o números en un formato especificado.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia de entrada/salida	LPRINT USING	Envía a la impresora cadenas o números en un formato especificado.	x	o	o
	OPEN	Realiza el proceso de apertura de ficheros.	o	o	o
	CLOSE	Realiza proceso de cierre de ficheros.	o	o	o
	INPUT#	Lee datos de un fichero, y los asigna a una variable.	o	o	o
	PRINT#	Saca a un fichero especificado el resultado evaluado de expresiones.	o	o	o
	PRINT# USING	Saca a un fichero en un formato especificado el resultado evaluado de expresiones.	o	o	o
	LINE INPUT#	Lee una línea completa de un fichero especificado y lo asigna a una variable cadena ignorando los caracteres delimitadores.	o	o	o
	FIELD	Asigna áreas de variables en un buffer para un fichero directo.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Sentencia de Entrada/Salida	GET	Lee un registro especificado y lo carga en un buffer desde un fichero directo.	o	o	o
	PUT	Escribe el contenido de un buffer en un fichero directo especificado.	o	o	o
	DSKO\$	Escribe el contenido de un buffer directo del sistema en un sector especificado.	o	o	o
Control de Pantalla Función Gráfica	WIDTH	Especifica el número de líneas y el número de columnas en pantalla.	o	o	o
	CONSOLE	Especifica el tamaño de ventana de scrolling.	o	o	o
	COLOR	Especifica el color visualizado de caracteres y gráficos así como el color de fondo.	o	o	o
	COLOR	Especifica un código de paleta.	x	x	o
	SCREEN	Especifica un código VRAM activa y código VRAM de visualizada.	x	x	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Control de Pantalla. Función Gráfica.	CLS	Borra una pantalla especificada y mueve el cursor a la posición superior izquierda.	o	o	o
	LOCATE	Mueve el cursor a una posición especificada en pantalla.	o	o	o
	PSET	Pone un punto en una posición especificada en pantalla.	o	o	o
	PRESET	Visualiza un punto en una posición especificada, en el color del fondo.	o	o	o
	LINE	Visualiza una línea o un rectángulo utilizando puntos o caracteres.	o	o	o
	CONNECT	Conecta puntos de coordenadas especificadas mediante líneas rectas.	o	o	o
	SYMBOL	Visualiza una cadena de tamaño, posición y orientación especificados.	o	o	o
	GET@	Lee caracteres o series de puntos de un área especificada de pantalla y los guarda en una matriz.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Control de Pantalla Función Gráfica	PUT@	Visualiza caracteres o una serie de puntos leídos por GET@ en una posición especificada.	o	o	o
	CIRCLE	Dibuja un círculo o arco circular con centro en una posición especificada.	o	o	o
	GCURSOR	Lee coordenadas de punto del cursor gráfico desde la pantalla.	o	o	o
	PAINT	Pinta (rellena) áreas limitadas por los colores especificados.	o	o	o
Función Tocar Música	PLAY	Toca música.	x	x	o
	SOUND	Controla el generador de sonido programable (PSG).	x	x	o
Función de tecla programable	KEY	Asigna una cadena de caracteres a una tecla de función programable.	o	o	o
	KEY LIST	Visualiza una cadena de caracteres asignada a las teclas de función programable.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función de tecla programable	KEY(n) ON/OFF/ STOP	Habilita, inhibe o aplaza una interrupción desde una tecla de función programable.	o	o	o
	ON KEY(n) GOSUB	Define una rutina de interrupción de tecla de función.	o	o	o
Función de Interruptor Temporizador	ON TIME GOSUB	Define una rutina de interrupción del reloj.	o	o	o
	TIME	Establece el tiempo de interrupción del reloj.	o	o	o
	TIME ON/ OFF/STOP	Habilita, inhibe o aplaza interrupciones del reloj.	o	o	o
	ON INTERVAL GOSUB	Define una rutina de interrupción del reloj a intervalos.	o	o	o
	INTERVAL	Establece el intervalo de interrupción del reloj a intervalos.	o	o	o
	INTERVAL ON/OFF/ STOP	Habilita, inhibe o aplaza interrupciones del reloj a intervalos.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función Control de línea	ON COM(n) GOSUB	Define una rutina de proceso de la interrupción llamada por una interrupción de entrada desde una línea de comunicaciones.	o	o	o
	COM(n) ON/OFF/ STOP	Habilita, inhibe o aplaza interrupciones de entrada desde una línea de comunicaciones.	o	o	o
	OPEN	Abre un fichero para permitir la entrada o salida a través de una línea de comunicaciones.	o	o	o
	CLOSE	Cierra ficheros asignados a enlaces de comunicaciones.	o	o	o
	INPUT #	Introduce datos desde una línea de comunicaciones.	o	o	o
	LINE INPUT #	Introduce una línea de datos desde una línea de comunicaciones.	o	o	o
	PRINT #	Saca datos en un enlace de comunicaciones.	o	o	o
	LIST	Saca un listado de programa en una línea de comunicaciones.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función Aritmética	ABS	Devuelve el valor absoluto del argumento.	o	o	o
	ATN	Devuelve el arcotangente del argumento.	o	o	o
	COS	Devuelve el coseno del argumento.	o	o	o
	EXP	Devuelve e elevada a la potencia del argumento.	o	o	o
	FIX	Devuelve la parte entera del argumento.	o	o	o
	INT	Devuelve el mayor entero que sea menor o igual que el argumento.	o	o	o
	LOG	Devuelve el logaritmo natural del argumento.	o	o	o
	RND	Genera un número aleatorio entre cero y uno.	o	o	o
	SGN	Devuelve +1 si el argumento es positivo, 0 si el argumento es cero y -1 si el argumento es negativo.	o	o	o
	SIN	Devuelve el seno del argumento.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función Aritmética	SQR	Devuelve la raíz cuadrada del argumento.	o	o	o
	TAN	Devuelve la tangente del argumento.	o	o	o
	CSNG	Convierte el valor de una expresión en un número real de simple precisión.	o	o	o
	CDBL	Convierte el valor de una expresión en un número real de doble precisión.	o	o	o
	CINT	Convierte un número en entero, redondeando la parte fraccionaria.	o	o	o
Función de cadena	CHR\$	Devuelve el carácter que corresponde al valor del código de caracteres del argumento.	o	o	o
	HEX\$	Convierte un entero en su número hexadecimal equivalente.	o	o	o
	LEFT\$	Devuelve un número especificado de caracteres de la izquierda, de la cadena especificada.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función de cadena	MID\$	Devuelve el número especificado de caracteres de la cadena especificada, a partir del carácter identificado por la posición del carácter.	o	o	o
	OCT\$	Convierte un entero a un número octal equivalente.	o	o	o
	RIGHT\$	Devuelve el número especificado de caracteres de la derecha de la cadena especificada.	o	o	o
	SPACE\$	Devuelve el número especificado de espacios.	o	o	o
	STR\$	Devuelve una representación en cadena del argumento.	o	o	o
	STRING\$	Devuelve una cadena de caracteres consistente en el carácter especificado.	o	o	o
	ASC	Devuelve el código de carácter correspondiente al primer carácter de la cadena especificada.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función de cadena	INSTR	Devuelve la posición de carácter desde la cual empiezan los caracteres en la cadena especificada.	o	o	o
	LEN	Devuelve la longitud de la cadena especificada.	o	o	o
	VAL	Devuelve una representación numérica de la cadena.	o	o	o
Función General	CSRLIN	Devuelve la posición vertical actual del cursor.	o	o	o
	POS	Devuelve la posición horizontal actual del cursor y de la cabeza de impresión.	o	o	o
	LPOS	Devuelve la posición horizontal actual de la cabeza de impresión.	x	o	o
	POINT	Indica si un punto está visualizado en la posición especificada.	o	o	o
	ERR	Devuelve un número de error al producirse uno.	o	o	o
	ERL	Devuelve el número de línea donde se produjo el error.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función General	VARPTR	Devuelve la dirección en la cual se almacenan los datos identificados por la variable.	o	o	o
	USR	Llama a una rutina en lenguaje máquina por el argumento especificado.	o	o	o
	PEEK	Devuelve el contenido de la dirección de la memoria principal especificada.	o	o	o
	FRE	Devuelve el número de bytes sin usar en memoria.	o	o	o
	TIME\$	Devuelve la hora actual del reloj incorporado.	o	o	o
	TIME	Indica la hora en segundos desde medianoche.	o	o	o
	DATE\$	Devuelve la fecha actual del reloj.	o	o	o
	DATE	Indica la fecha actual en días desde 1 de enero.	o	o	o
	SPC	Imprime el número especificado de espacios.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función General	TAB	Imprime espacios hasta la posición de columna especificada.	o	o	o
	SCREEN	Devuelve el código de carácter o los atributos del carácter en la posición especificada.	o	o	o
Función de entrada/salida	CVI	Convierte una cadena de caracteres de 2 bytes especificada en datos numéricos equivalentes.	o	o	o
	CVS	Convierte una cadena de caracteres de 4 bytes especificada en datos numéricos equivalentes.	o	o	o
	CVD	Convierte una cadena de caracteres de 8 bytes especificada en datos numéricos equivalentes.	o	o	o
	MKI\$	Convierte un entero en una cadena de caracteres equivalente.	o	o	o
	MKS\$	Convierte un número de simple precisión en una cadena de caracteres equivalente.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función de entrada/salida	MKD\$	Convierte un número de doble precisión en una cadena de caracteres equivalente.	o	o	o
	EOF	Devuelve un valor que indica la condición EOF (fin de fichero) o no EOF.	o	o	o
	LOF	Devuelve el número de caracteres almacenados en el buffer de entrada.	o	o	o
	LOC	Da el número de registro en el fichero de acceso directo especificado al cual se va a acceder con la siguiente sentencia GET o PUT.	o	o	o
	DSKI\$	Trae el contenido del sector especificado al buffer de acceso directo del sistema.	o	o	o
	DSKF	Devuelve el número de grupos de registros disponibles en el disco especificado.	o	o	o

o: Utilizable
x: No utilizable

Tipo	Sentencia	Significado	Versión		
			V1.0	V2.0	V3.0
Función de entrada/salida	INPUT\$	Introduce una cadena formada por el número especificado de caracteres desde el teclado o el fichero especificado.	o	o	o
	INKEY\$	Devuelve el carácter correspondiente a la tecla que se pulsa a continuación, y un carácter nulo si no se pulsa ninguna tecla.	o	o	o

		1234567890			
1234567890	1234567890	1234567890			
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

1234567890