

MBC - 1000

SBASIC II

MANUAL DE PROGRAMACION

VERSION 2.0

Traducción realizada
por NANOSOFT
1.983



SANYO

INDICE

Introducción	1
Símbolos para la Descripción Sintática	2
(1) MODOS DE OPERACIÓN DE SBASIC	3
1.1 Inicialización	3
1.2 Signo del Sbasic y Cursor	4
1.3 Modo Directo	4
1.4 Modo de Edición del Programa	5
1.5 Modo de Ejecución	5
1.6 Retroceso al Modo del Sistema Operativo	5
(2) LINEA Y SENTENCIAS DEL PROGRAMA	6
2.1 Caracteres y Símbolos usados en los Programas	6
2.2 Líneas y Números de Línea	7
2.3 Palabras clave y espacios	8
(3) DATOS	8
3.1 Datos	8
3.2 Constantes	9
3.3 Variables	10
(4) EXPRESIONES Y OPERADORES	11
4.1 Expresiones y Operadores	11
4.2 Operadores aritméticos	11
4.3 Expresiones aritméticas	11
4.4 Operadores de relación	12
4.5 Expresiones de relación	13
4.6 Operadores lógicos	14
4.7 Expresiones lógicas	14
4.8 Evaluación de Expresiones Combinadas	17
4.9 Expresiones de strings (cadenas alfanuméricas)	17
(5) PALABRAS DE INSTRUCCIÓN	18
(6) FUNCIONES	77
(7) EDICIÓN DEL PROGRAMA	87
7.1 Edición del Programa	87
7.2 Edición de Línea	87
7.3 Corrección de Línea	89

(8)FICHERO DE DISCOS	90
8.1 Clases de Ficheros	90
8.2 Nombres de Ficheros	90
8.3 Números de Ficheros	92
8.4 Instrucciones para los ficheros	92
8.5 Ficheros de Programas	93
8.6 Ficheros de Datos	93
8.7 Ficheros secuenciales	93
8.8 Ficheros directos	95
8.9 Entradas y Salidas de Ficheros e Imágenes de Discos	96
8.10 Números de Ficheros de un Diskette.	99

Apéndice

(A) Tabla de Códigos de Caracteres	100
(B) Mensajes de error	101
(C) Mapa de la Memoria	102
(D) Lista de Palabras de Instrucción por su función	105
(E) Lista de Funciones	108
(F) Lista de Palabras reservadas	109

INTRODUCCION

SBASICII es un intérprete de BASIC, especialmente, diseñado para el mini-computador de sobremesa MBC-1000. Ofrece una amplia gama de aplicaciones en el campo del trabajo administrativo, así como en el cálculo técnico, gracias a su gran cantidad de instrucciones, funciones matemáticas, amplia área del usuario y función de edición de líneas.

Por favor, lea el manual de programación detalladamente y use el SBASICII de manera eficaz para su programación. En este manual, únicamente, se describen las operaciones, programas, palabras de instrucción y edición de líneas del SBASICII. En lo referente al funcionamiento del MBC-1000, consulte su manual de instrucciones.

Símbolos para una Descripción Sintática

1. **t]** Las indicaciones entre corchetes, pueden ser omitidas en las descripciones de los programas. Cuando se omiten, las operaciones son diferentes, de acuerdo con las respectivas palabras de instrucción, por lo que se sugiere una referencia para cada palabra.

2. **{ }** Las indicaciones entre llaves, pueden ser bien omitidas o repetidas en cada ocasión.

3. **<>** Las indicaciones entre corchetes de ángulo, no deberán omitirse, sino que se escribirán de forma voluntaria, siempre y cuando pertenezcan a sus definiciones concretas.

4. **| (raya vertical)** Las indicaciones separadas mediante una raya vertical son mutuamente exclusivas; deberá escogerse una sola indicación. Cuando aparecen de forma indefinida, las dos indicaciones opcionales se subrayarán.

5. **Caracteres y Símbolos**
Todos los caracteres y símbolos que aparecen en las sentencias generales y comandos poseen un significado específico. Deben escribirse tal y como son.

- (Menor que)
- (Mayor que)
- = (Símbolo de asignación o de igual)
- ? (Punto de interrogación)
- @ (Arroba)
- [(Corchete izquierdo)
-] (Corchete derecho)
- \ (Barra invertida o símbolo de división entera)
- ^ (Símbolo de exponentiación)
- (Subrayado)
- { (Llave izquierda)
- } (Llave derecha)
- ~ (Onda o signo "hacia")
- (Señal de tabulación)

Referente a los caracteres especiales, consultar el Apéndice A "Tabla de los Códigos de Caracteres".

Tanto los comandos como las especificaciones de los programas se describen con letras mayúsculas y minúsculas, así como mediante símbolos reconocidos sintáticamente. Las letras mayúsculas y minúsculas se usan y son interpretadas para describir los comandos y especificaciones, sin embargo, las letras minúsculas usadas para los datos literales, se almacenan como teclas.

(1) MODOS DE FUNCIONAMIENTO DE SBASIC

1.1 Inicialización

La siguiente línea debe escribirse en el Modo del Sistema Operativo, para inicializar el interpretador SBASIC.

A> SBASIC[<nombre del fichero>][/F:<número de ficheros de datos que va a usarse>]
[/M:<dirección máxima>]<CR>

Una vez se ha procesado la línea, el interpretador se carga en memoria y el control entra en el Modo BASIC.

- <nombre del fichero>

Especifica el nombre del fichero de disco en el cual está almacenado el programa. Una vez especificado, los programas que deben ejecutarse después de la inicialización del intérprete BASIC, son cargados automáticamente y ejecutados después de la inicialización. En el caso que se omita el nombre del fichero, el control entra en el modo SBASIC pero espera que se teclee una vez aparecido en la pantalla el signo de BASIC. En lo referente a los ficheros de los programas que incluyen los nombres de ficheros, ver (8) "FICHEROS DE DISCOS".

- <número de ficheros de datos que han de usarse>

Define la cantidad máxima de ficheros de datos que, un programa con el nombre del fichero especificado, puede abrir cada vez. Cuando esto queda omitido, se considera que el número es el de "3".

En SBASIC, el número máximo de ficheros que pueden abrirse cada vez, es de 15. (Se reserva un buffer al especificar el número de ficheros simultáneos. Ver 8.3 "Número de Ficheros").

- <dirección máxima>

Establece el fin de la memoria, cuando se carga un programa BASIC. Es necesario especificar la < dirección máxima> cuando el usuario desea definir el tamaño de la memoria usado en el BASIC, con objeto de insertar subrutinas en lenguaje assembler. Cuando se omite la < dirección máxima> en el momento de la inicialización, el sistema define la dirección más alta posible y hace aparecer por la pantalla, el número de bytes que se usan para el área de usuario.

Ejemplo: SBASIC<CR>
SBASIC^"PROGRAMA-1" <CR>
SBASIC^"PRGM-1" /F:5<CR>
SBASIC^"PRGM-1" /F:7/M:&HCOOO<CR>

1.2

EL SIGNO DE BASIC Y CURSOR

El modo BASIC es un estado tal que el sistema queda a la espera de que se tecleen instrucciones BASIC. Se pueden entrar nuevas instrucciones una vez se ha ejecutado la anterior. Cuando se está en modo BASIC, aparece la señal (>).

Ready

>

Símbolo de subrayado (destelleante)

Cuando aparece la señal de BASIC, puede procederse con una instrucción o programa de BASIC.

1.3 Modo Directo

Cuando aparece una señal de BASIC y se entra una sentencia del programa, sin antes un número de línea y con <cr> al final, la sentencia se ejecuta inmediatamente. Esto se llama el Modo Directo. El resultado de la operación de escribir la sentencia en Modo Directo, queda almacenado en la memoria, sin embargo su instrucción queda borrada. También es posible la ejecución de entradas múltiples en Modo Directo.

(ver 2.2 "Líneas y Números de Línea")

Ejemplo: > A=3.14*5.8^2:PRINT A<CR>
105.63

> FOR I=1 TO 10:PRINT I:NEXT I<CR>
1
2
3
4
5
6
7
8
9
10

10 FOR I=1 To 100
20 Print I;" ";
30 NEXT I

(Δ = Espacio)

1.4

Modo de Edición de Programas

El Modo de Edición de Programas se emplea para entrar, editar y almacenar en la memoria, los programas de BASIC. (En este apartado sólo se incluyen generalidades, para una información más detallada sobre la edición de programas, consultar (7) "EDICION DE PROGRAMAS".)

Cuando se entra un número de línea y luego se escribe una sentencia, esta se carga en la memoria, como una unidad del programa fuente. El número de línea se lleva a cabo de manera automática mediante el uso del comando AUTO (ver 5.1), la operación de la reenumeración para modificar simultáneamente todos los números de línea de los programas fuente, también se lleva a cabo a través del comando RENUM (ver 5.60). La sucesión de "número de línea, sentencia" se va escribiendo repetidamente, para que los programas de origen queden almacenados en secuencia en la memoria para un oportuno uso mediante el comando RUN (ver 5.63).

1.5

Modo de Ejecución

Los programas fuente, llevados a cabo en el Modo de Edición de Programas y almacenados en la memoria, son ejecutados en secuencia en que están los números de línea, mediante el comando RUN. El proceso puede ser detenido, pulsando la tecla de BREAK o mediante el STOP en el programa. Puede ser continuado usando la sentencia de CONT (ver 5.8). Si durante el estado de HALT se llevan a cabo modificaciones en el programa fuente o en caso de producirse algún error, mediante la ejecución directa, el programa no podrá continuarse. El modo de Ejecución retrocede al de basic, cuando se lleva a cabo la sentencia de END (ver 5.15), durante la ejecución del programa o cuando el programa fuente se ha acabado (no existen sentencias ejecutables). Sin embargo, cuando se usa el comando SYSTEM en modo directo o en programa, el control retrocede al Modo del Sistema Operativo.

1.6

Retroceso al Modo del Sistema Operativo

El control retrocede desde el modo BASIC al modo del Sistema Operativo, cuando SYSTEM aparece en el programa el comando SYSTEM, durante la ejecución o cuando el comando SYSTEM se lleva a cabo de forma directa en el modo de BASIC. Cuando se pulsa la tecla CTL y la de RESET simultáneamente, el control retrocede al Modo del Sistema Operativo en el momento de que se instala un disco de sistema en la unidad A y en caso de no ser así, el sistema pasa al estado de ESPERA, hasta que se instala el disco. Para el cambio del Modo del Sistema Operativo al de BASIC, es necesario el procedimiento de inicialización descrito en el apartado 1.1. Cuando se vuelve a realizar la inicialización de BASIC, puede modificarse tanto los parámetros, como el número de ficheros de datos a usar y la dirección máxima que ha sido especificada ,en el momento de la inicialización.)

(2) LINEA Y SENTENCIAS DE PROGRAMAS

2.1 Caracteres y Símbolos usados en los Programas

Estos son los caracteres y símbolos usados en BASIC:

- o 26 letras mayúsculas
- o 26 letras minúsculas
- o 10 números
- o 32 símbolos
 - ! (Admiración)
 - " (Comillas)
 - # (Signo de números)
 - \$ (Signo del dólar)
 - % (Tanto por ciento)
 - & (Ampersand)
 - ' (Apóstrofe)
 - ((Paréntesis izquierdo)
 - * (asterisco)
 - + (Signo de más)
 - (Signo de menos)
 - , (Coma)
 - . (Punto o señal de decimal)
 - / (Símbolo de división o barra)
 - : (Dos puntos)

ABC,,,	
abcd...	
0123456789	
(espacio)	
;	(Punto y coma)
<	(signo de menor que)
>	(signo de mayor que)
=	(señal de igual o de asignación)
?	(Interrogación)
@	
[(corchete izquierdo)
]	(corchete derecho)
\	(símbolo de división)
\	(barra invertida)
_	(Subrayado)
{	Llave izquierda
}	Llave derecha
~	

Estos caracteres y símbolos se convierten en mayúsculas cuando se almacenan en la memoria. Si las letras de la parte inferior del teclado se consideran como strings (datos literales), quedan almacenadas tal y como son.

Los caracteres se escriben en el programa pulsando la correspondiente tecla en el teclado. Los caracteres especiales se consiguen mediante el uso múltiple de las teclas. En los programas BASIC, los caracteres especiales de la columna 0 y 1 de la tabla de códigos de caracteres incluida al final de este manual, no pueden entrarse a través del teclado. Se escriben en los programas exclusivamente mediante el uso de las funciones CHR\$.

2.2

Líneas y Números de Línea

Una Línea es la unidad mínima de un programa, como elemento consistente en una línea, un grupo de sentencias (descripción de instrucciones) y un <CR>(para acabar una Línea).

El formato normal de una Línea es el siguiente:

< número de Línea> < sentencia> {<sentencia>} <CR>

* Un número de Línea es un entero positivo de hasta 5 dígitos, en un rango de 0 a 65535. (Se asignan 2 bytes para cada número de Línea en la memoria.) Los ceros a la izquierda quedan ignorados.

En un parámetro numérico, no se incluyen espacios para mostrar un número de Línea.

Ejemplo:	00100	→	Igual a 100
	1.0,0	→	Erróneo

El número de Línea nos indica el índice para el control de operación del programa y al mismo tiempo, para averiguar la Línea para la oportuna edición del programa.

- < Sentencia> es una instrucción escrita según las normas sintáticas de BASIC y también es la unidad mínima del elemento para componer las instrucciones. El capítulo 5 nos explica como escribir cada instrucción.
- Dos puntos (:) es un delimitador localizado entre dos sentencias, al llevar a cabo una serie de sentencias en una única Línea de programa. Una Línea de programa compuesta por una sola sentencia se llama "Sentencia Simple" y la que se compone de varias sentencias ,las cuales están delimitadas por dos puntos, se llama "Sentencia Múltiple".
- <CR>nos indica el final de una Línea lógica y se entra mediante la tecla<CR>.

La longitud de una Línea no debe pasar de 255 dígitos incluyendo el número de Línea y los espacios.

2.3

Palabras y espacios

Los comandos, las sentencias y nombres de función son las palabras, especialmente caracterizadas en SBASIC y los espacios no se incluyen. Por consiguiente, tanto las palabras como las variables, constantes y operadores lógicos deben ser delimitados por espacios, paréntesis, operadores numéricos y otros delimitadores reconocidos sintáticamente. En cualquier otro caso, los espacios se pueden usar libremente y quedan ignorados, siempre que existan espacios en las sentencias.

Ejemplo:	PRINT	Erróneo
	A = B + C	Válido
	A=B AND C	A se considera como una multiplicación lógica de B y C
	A=BANDC	A la variable A se le asigna el contenido de la variable "BANDC"

(3) DATOS

3.1 Datos

Para el procesamiento del programa, los datos se escriben mediante números o caracteres. Los datos con números son considerados como "datos numéricos" y lo mismo referente a los caracteres, se les llama "strings". Para los datos, se pueden asignar tanto una constante, como una variable. Hay dos tipos de variables: variables simples y variables suscritas (matrices). Los datos numéricos se dividen en tres categorías a saber: enteros, números reales de precisión simple y números reales de precisión doble.

La siguiente tabla nos muestra un ejemplo de los datos.

		String	Entero	Datos Numéricos		
				Número real de precisión simple	Número real de precisión doble	Número Octal y Hexadecimal
Constantes	"Tokyo"	2365		365.13 -9876E35 -123.456!	-12345678.9 .36789D16 12.5678#	&7354 &H3AF5
Variables simples	A\$	ABC%	XY!	D#		
	B\$(3,4,5)	X%(N)	Z!(I,J)	DB#(1Q)		
(Memoria)	1 byte para ch.	2 bytes	4 bytes	8 bytes		

3.2 Constantes

(1) Constantes de enteros

Números entre -32768 y +32767 sin decimales ni fracciones

Ejemplo:

365
-12345

(2) Números reales (Precisión simple) números significativos de 6 dígitos con un exponente entre -64 y +63 expresado de estas tres maneras:

(i) Número hasta 6 dígitos expresado como \pm XXXX.XX

Ejemplo: 1234.56

(ii) Número con un exponente (E) expresado como \pm . XXXXXXE \pm XX

Ejemplo: 123456E-05

(iii) Número con un sufijo "!" escrito \pm XXX.XXX!

Ejemplo: 123!

(3) Constantes de precisión doble (números reales de doble precisión) números significativos de 14 dígitos con un exponente entre -64 y +63 expresado de las 3 maneras siguientes:

(i) Número de hasta 14 dígitos expresado como \pm XXXXXXXXXXXX.XXXX

(ii) Número con un exponente expresado como \pm .XXXXXXXXXXXXXXXXD \pm XX

(iii) Número con un sufijo "#" escrito \pm XX.XXXX#

(4) Números octales

El rango de los números octales es de 0 a 177777 con un prefijo "&" o "&O"

Ejemplo: &175432
&0324

(5) Números hexadecimales

Los números hexadecimales oscilan entre 0 y FFFF con un prefijo "&H"

Ejemplo: &H256
&HA4F5

(6) Constante de caracteres

Los caracteres han de estar entre comillas ("") dentro de 255 dígitos

Ejemplo: "ABC"
"XY-123Lot"

No se pueden usar comillas dentro de los strings.

3.3 Variables

(1) Nombre de la variable

Las variables son los valores cambiables de un programa y quedan definidas al especificar un nombre para cada una. Los nombres de las variables se escriben en sucesión alfanumérica, con una longitud opcional, empezando por un carácter alfanumérico. Los primeros dos caracteres muestran la variable.

Ejemplo:

TR
TRE

TRANS



Se considera como la misma variable

Los nombres de las variables no han de coincidir con las instrucciones del BASIC, aunque pueden incluir las mismas letras. Los nombres de las variables empezando por FN o USR son falsas.

Ejemplo:

AND=100	Falso
BAND=0	Aceptable
AFN10=A+B	Aceptable
FNABC=1	Falso
BAFN=1	Aceptable

(2). Tipos de variables

Las variables se clasifican en strings o numéricas (enteros, números reales de precisión simple y números reales de precisión doble).

El tipo de datos se define por la sentencia DEF (ver 5.11) o por un sufijo al final de la variable, como un símbolo de especificación.

Tipo de variable		Símbolo	Ejemplo
Numéricos	Entero	%	A%, INTEGER%
	Número real de precisión simple	!	R0,REAL!
	Número real de precisión doble	#	B5#, DBL#
Caracteres	(String)	\$	LITERAL \$

Cuando una variable no tiene ningún símbolo de especificación del tipo a que corresponde, ni tampoco queda especificado por la sentencia DEF, se tomará un número real de precisión simple. (En defecto de la especificación del tipo para un nombre de variable numérica es de precisión simple)

(3) Matrices (Variables de subíndices)

Las matrices se especifican mediante la sentencia DIM (ver 5.14) antes de ser usadas. No existe ningún tipo de limitación en cuanto al tamaño de la matriz ni a la cantidad de elementos, sin embargo, la longitud total debe ser aceptable por la capacidad máxima de memoria de la máquina.

Cuando una matriz tiene 10 o menos elementos, se puede usar incluso sin haberla definido previamente mediante la sentencia DIM. (La declaración DIM se asume automáticamente al usar la variable por primera vez. Ver 5.14).

(4) EXPRESIONES Y OPERADORES

4.1 Expresiones y Operadores

Un operador es un símbolo que muestra el método opcional de añadición, para reducir o comparar datos y una combinación en que los números de los elementos de datos, de acuerdo con las normas sintácticas, se llama expresión. Una expresión ofrece un valor resultante de la operación de datos efectuada según el orden que especifiquen los operadores. En este manual, a este valor le llamaremos resultado de evaluación, el cual podrá ser, naturalmente, tanto de datos de sucesión como numéricos.

4.2 Operadores Aritméticos

Los operadores aritméticos están divididos en los siguientes ocho tipos:

1 ^	Método de poder	A^B	Elevar A a B
2 -	Signo de menos	-A	Menos A
3 *	Multiplicación	$A*B$	Multiplicar A por B
4 /	División	A/B	Dividir A por B
5 \	División de enteros	$A\B$	Dividir A por B y eliminar los decimales
6 MOD	Resto	$A \text{ MOD } B$	Resto de A dividido por B Tanto A como B se redondean a enteros antes de efectuar la división
7 +	Suma	$A+B$	Añadir A y B
8 -	Resta	$A-B$	Restar B de A

4.3 Expresión Aritmética

Una expresión aritmética es una combinación de elementos de datos numéricos, mediante operadores aritméticos y generalmente se escriben así:

<elemento del dato> {<operador aritmético>} <elemento de datos>}

- Los elementos de datos se clasifican de esta manera:
 1. Constante numérica
 2. Variable numérica
 3. Función numérica
 4. (Expresión aritmética) Las expresiones aritméticas van entre paréntesis.
- La operación de una expresión aritmética, se lleva a cabo con la siguiente prioridad, cuyo resultado de evaluación se convertirá en un valor numérico.
 - 1) Cálculo de la expresión incluida
 - 2) Cálculo del poder
 - 3) Signo de menos
 - 4) Multiplicación y división
 - 5) División
 - 6) Resto
 - 7) Suma y resta

Los cálculos que tienen la misma prioridad de operación, deben hacerse de izquierda a derecha.

Ejemplo:

$$Z * (Z+1)^2 / 3 + 1 - 2 \rightarrow 5$$

4.4 Operadores de Relación

Los operadores de relación dirigen la operación para referir o comparar un elemento de dato con otro. Se dividen en los siguientes seis tipos:

1) =	Igual?	$A=B$	Es A igual a B?
2) <>	Diferente?	$A \neq B$	Es A diferente a B?
3) <	Menor?	$A < B$	Es A menor que B?
4) >	Mayor?	$A > B$	Es A mayor que B?
5) \leq	Igual o menor?	$A \leq B$	Es A igual o menor que B?
6) \geq	Igual o mayor?	$A \geq B$	Es A igual a o mayor que B?

4.5 Expresiones de Relación

Una expresión de relación es una combinación de elementos de datos a través de operadores de relación y normalmente se escriben de la siguiente manera:

< elemento de datos > {operador de relación} < elemento de datos >

- Un elemento de datos es una expresión aritmética o bien una sucesión de caracteres. Los elementos de datos que pueden escribirse en una expresión de relación son del mismo tipo: expresiones aritméticas o sucesiones de caracteres. (Por ejemplo, las expresiones aritméticas no pueden compararse con los datos de sucesión o vice versa).
- Cuando los elementos de datos están interrelacionados mediante operadores de relación, el resultado de evaluación se convertirá en -1 si la expresión se confirma (cierta) y en 0 si no se confirma (falsa).
- La comparación de los valores numéricos se hace de dos maneras, dependiendo de si un valor es menor o mayor que el otro.
- El tamaño determina la comparación de los valores numéricos, en otras palabras, la comparación de los valores numéricos se lleva a cabo determinando si el valor es menor o mayor que el otro.
- La comparación de caracteres (dos strings), se lleva a cabo mediante el siguiente procedimiento:
 - 1) El tamaño de un carácter se determina de acuerdo con la tabla de códigos de caracteres del Apéndice A.
 - 2) El tamaño de las sucesiones que tienen más de dos caracteres, se comparan de la siguiente manera:
 - La comparación del tamaño basada en la norma 1), debe hacerse en el primer par de caracteres, con el segundo, etc.
 - La sucesión que empieza por un carácter de tamaño menor, se determina como menor.
 - Tanto si una sucesión acaba habiéndose hecho la determinación como no, la más corta se considerará como la más pequeña.

Ejemplo:

"A">"B"	-1
"?"<"="	0
"ABC"<"ABCA"	-1
"ABA">"AA"	-1
"XYZ"<"ABC-1"	0

- En una expresión, se incluyen más de dos operadores de relación, la operación se hace de izquierda a derecha para permitir un resultado de evaluación.

4.6 Operadores Lógicos

Estos seis operadores lógicos siguen un orden de prioridad opcional.

1) NOT	No aceptado (inversión)
2) AND	Producto lógico
3) OR	Suma lógica
4) XOR	Suma lógica exclusiva
5) IMP	Implicación
6) EQV	Equivalente

4.7 Expresiones lógicas

Normalmente, una expresión lógica tiene el siguiente formato:

{NOT} <elemento de datos>{operador lógico} {NOT} <elemento de datos>}

- Para los elementos de datos, se pueden usar tanto las expresiones aritméticas, como las expresiones aritméticas. En el caso de ser una expresión aritmética, su resultado de evaluación deberá ser un entero entre -32768 y +32767. Si el resultado de evaluación fuese un número real, sería redondeado. Si sobrepasa este rango, aparecerá un error.
- De una expresión lógica, viene el resultado de evaluación con el siguiente procedimiento:
 - Cuando se incluyen más de dos operadores lógicos:
 - Ejecutar el procesamiento, según las prioridades.
 - Si los dos operadores lógicos diferentes tienen la misma prioridad, se ejecuta de izquierda a derecha.
 - Cuando el valor de <datos> se le asigna -1 (verdad) ó 0 (falso), el resultado de evaluación de la expresión lógica será de -1 ó 0.
 - Cuando se incluye un entero en el resultado de evaluación de <elementos de datos>, la operación lógica se desarrolla con anotaciones binarias (existen binarios enteros para cada bit), considerando al resultado como un número binario. El resultado de evaluación se convertirá en un entero.

Ejemplo:

1) Definición de los operadores lógicos:

o NOT	X	NOTΔX	
	1	0	
	0	1	
o AND	X	Y	XΔANDΔY
	1	1	1
	1	0	0
	0	1	0
	0	0	0

o OR	X	Y	$X \Delta \text{OR} \Delta Y$
	1	1	1
	1	0	1
	0	1	1
	0	0	0
o XOR	X	Y	$X \Delta \text{XOR} \Delta Y$
	1	1	0
	1	0	1
	0	1	1
	0	0	0
o IMP	X	Y	$X \Delta \text{IMP} \Delta Y$
	1	1	1
	1	0	0
	0	1	1
	0	0	1
o EQV	X	Y	$X \Delta \text{EQV} \Delta Y$
	1	1	1
	1	0	0
	0	1	0
	0	0	1

2) Operación lógica incluyendo entero(s)

- o $15 \Delta \text{AND} \Delta 16 \rightarrow 0$

$$\begin{array}{rcl} 15 & \rightarrow & 1111 \\ 16 & \rightarrow & 10000 \\ \hline 15 \Delta \text{AND} \Delta 16 & \rightarrow & 00000 \end{array} \rightarrow 0$$

- o $15 \Delta \text{AND} \Delta 10 \rightarrow 10$

$$\begin{array}{rcl} 15 & \rightarrow & 1111 \\ 10 & \rightarrow & 1010 \\ \hline 15 \Delta \text{AND} \Delta 10 & \rightarrow & 1010 \end{array} \rightarrow 10$$

- o $-1 \Delta \text{AND} \Delta 16 \rightarrow 16$

$$\begin{array}{l} -1 \rightarrow 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ 16 \rightarrow 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

$$-1 \Delta \text{AND} \Delta 16 \rightarrow 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \rightarrow 16$$

- o $8 \Delta \text{OR} \Delta 16 \rightarrow 24$

$$\begin{array}{rcl} 8 & \rightarrow & 1000 \\ 16 & \rightarrow & 10000 \\ \hline 8 \Delta \text{OR} \Delta 16 & \rightarrow & 11000 \rightarrow 24 \end{array}$$

- o $15 \Delta \text{OR} \Delta 16 \rightarrow 31$

$$\begin{array}{rcl} 15 & \rightarrow & 1111 \\ 16 & \rightarrow & 10000 \\ \hline 15 \Delta \text{OR} \Delta 16 & \rightarrow & 11111 \rightarrow 31 \end{array}$$

4.8 Evaluación de las Expresiones Combinadas

Las expresiones de relación o aritméticas pueden usarse como elementos de datos en una expresión lógica. Por lo tanto, una combinación de diversas expresiones, se lleva a cabo siguiendo el orden de 1) expresión aritmética, 2) expresión relacional y 3) expresión lógica.

Ejemplo:

$$A \underbrace{+ B}_{1} = C \underbrace{+ D}_{1} \text{ OR } (A \underbrace{- B}_{1}) \underbrace{* C \text{ AND } D}_{2}$$

3 4
 5

Cuando A=1, B=2, C=2 y D=1, el resultado de evaluación de la anterior expresión será de -1.

Destaquemos que el "=" usado en la sentencia de LET (ver 5.33) y el "=" como un operador de relación lógica, son distintos en significado.

Ejemplo:

LET A=2=3	a 0 se le asigna A
LET A=B=1	a -1 se le asigna A cuando B= es 1
	a 0 se le asigna A cuando B≠1

4.9 Expresiones de Strings

Las expresiones de strings combinan una sucesión de caracteres con otra y se escriben siguiendo este formato:

⟨elemento de dato⟩ {+⟨elemento de dato⟩}

- Se puede usar, como elemento de datos, cualquiera de estos tres:
 - 1) Constante de cadena
 - 2) Variable de cadena
 - 3) Función que resulta en una cadena (Funciones de caracteres)
- El resultado de evaluación de una expresión de cadena también es una sucesión de caracteres con menos de 255 dígitos. Cuando la longitud de una sucesión, excede el límite, aparece un error.

Ejemplo:

"TO" + "KYO" → "TOKYO"

```
> 10ΔA=3
> 20ΔA$="FILE"
> 30ΔB$=A$ + "-" + RIGHTS$(STR$(A), 1)
> 40ΔPRINTAB$
```

> RUN

→ FILE-3

Saturn

(5) PALABRAS DE INSTRUCCION

Las palabras de instrucción de SBASIC efectuan ejecuciones directas y sirven para escribir programas de usuario. Generalmente, en SBASIC, no existe discriminación entre los comandos y las sentencias, sin embargo, mediante unas palabras de instrucción, el control no continua el proceso, retrocediendo al modo BASIC, incluso en el caso de que se haya terminado la ejecución.

Las palabras de instrucción se muestran en un orden alfabético pero en algunas palabras de instrucción interrelacionadas, no se sigue este orden de manera intencionada.

Las siguientes palabras de instrucción hacen que el control vuelva al modo de BASIC:

- AUTO
- DELETE
- END
- LIST/LLIST
- LOAD (Cuando no se especifica la opción R)
- MERGE
- NEW
- RENUM
- STOP (El programa puede resumir mediante el CONT.)

Una vez efectuada la ejecución, el control vuelve al Modo del Sistema Operativo, mediante la siguiente palabra de instrucción.

- SYSTEM

5.1 AUTO

El AUTO instruye al interpretador para que genere automáticamente los números de línea en el momento en que se está escribiendo el programa.

AUTO [< número linea > [, [< incremento >]]]

- Una vez especificado número de línea , el sistema muestra el número de línea especificado y queda en la espera de la siguiente sentencia para poder seguir. Cuando se teclea una sentencia y se ha pulsado <CR>, el sistema muestra otro número de línea mayor y espera otra sentencia. El mismo proceso se va repitiendo.
- Si no se especifica el número de línea, la creación de los números empieza desde el 10. Cuando no se hace ninguna especificación, el número será el 10. (Las líneas serán 10, 20, 30, etc.)
- Cuando se especifica el <número de línea> como número de línea , queda designado el incremento que se había usado justamente antes de la ejecución.

- El modo AUTO desaparece, pulsando la tecla de <BREAK> y el control vuelve al modo de BASIC. Cuando el número de línea sobrepasa el 65535 o cuando se encuentra un error de edición de línea en la sentencia que se ha escrito, aparece un mensaje de error y el control vuelve al modo de BASIC (ver Apéndice B.)
- Cuando se asigna 0 a <incremento>, se considera como un error.

Ejemplo:

```
> AUTO    100, 20
> 100.....
> 120.....
> AUTO.
```

(Aparecerá el último número de línea del programa que ha sido almacenado en la memoria)

5.2 BEEP

La instrucción BEEP genera un tono audible.

BEEP

Suena durante medio segundo.

5.3 CALL

La sentencia de CALL demanda las subrutinas escritas en el lenguaje Assembler del programa de BASIC. (Para llamar a las subrutinas escritas con BASIC, se emplea la sentencia de GOSUB (ver 5.22.))

CALL <dirección de arranque>] ,<variable> ,{<variable>}]

- <dirección de arranque> sirve para inicializar una subrutina de assembler y escrita con una constante o variable.
- <variable> es un argumento para transferir el procesamiento a la subrutina de assembler requerida y pueden escribirse hasta 3 variables.
- La transferencia a la subrutina mediante un argumento se realiza usando los registros CPU (Z-80A).

Dirección en la que se almacena la primera variable ... registro HL

Dirección en la que se almacena la segunda variable ... registro DE

Dirección en la que se almacena la tercera variable ... registro BC

Cuando se ha entrado la dirección en la que se ha almacenado la variable (argumento) especificada, en el correspondiente registro, el control va hasta la dirección de arranque de una subrutina.

- Cuando se ejecuta la instrucción de RET en un programa de subrutina, el control hace la transferencia a la sentencia a continuación del CALL del programa de BASIC, para el cual se ha pedido la subrutina.
- Vea más adelante la sentencia DEF USR.

Ejemplo:

CALL & HA000, A,B,C

Las direcciones A, B, y C son entradas en los registros HL, DE y BC respectivamente y entonces, el control va hasta la dirección A000 de una subrutina.

5.4 CHAIN/COMMON

El CHAIN desarrolla el siguiente procesamiento:

- Estando en un programa, carga otro diferente del disco.
- Transfiere el contenido y las variables del programa en proceso al último programa entrado.
- Ejecuta el último programa entrado.

Mediante el CHAIN, el programa principal puede dividirse en subprogramas y ser ejecutados continuamente

CHAIN [M _{ERGE}] <nombre fichero> [, [, <número linea 1>]] [, ALL]
[, DELETE <número linea 2> — <número linea 3>]]

- < nombre de fichero > aquí se especificará el del último programa que ha de entrarse.
- < número de línea 1 > es la línea de inicio del último programa entrado, del cual surgirá la ejecución del nuevo programa cuando así se especifique. Si se omite < el número de línea 1 >, la ejecución empezará en la primera línea de un subprograma.
- Las variables del programa actualmente ejecutado (programa precedente), son transferidas al último programa entrado (programa siguiente) siguiendo este proceso:
 - 1) Cuando las variables del programa precedente deban ser transferidas al siguiente se especifica la opción ALL, a través del CHAIN.
 - 2) Si las variables del programa precedente deben de ser transferidas sólo en parte, al siguiente programa; ALL queda omitido en la sentencia de CHAIN y sólo se especifican las variables necesarias mediante sentencias normales.

El formato de una sentencia COMMON es el siguiente:

COMMON <variable>[()]{<variable>[()]}

- <variable> en este caso, se ha de transferir al programa siguiente. Cuando se trata de una matriz, la especificación se escribe entre paréntesis como "A()".
- Cuando la sentencia de CHAIN, en la que se especifica la opción de ALL, se escribe para un programa en el que se incluye la sentencia de COMMON, se considerará como erróneo. Por otro lado, no se admite la sentencia de COMMON cuando ya existe en el programa, la sentencia de CHAIN, con la opción ALL.
- La sentencia de COMMON no se considera válida, sino se ha ejecutado la sentencia de CHAIN anteriormente. Un programa no puede incluir más de 11 sentencias de COMMON.
- Cuando se especifica el MERGE, la ejecución se lleva a cabo uniendo el programa precedente con el siguiente.
Es posible una superposición de las subrutinas, mediante la opción de MERGE. Es necesario especificar cual es la parte del programa precedente que se desea reemplazar por el siguiente. A este objeto la opción de DELETE es la más efectiva. En un formato normal, el número de línea 2 hasta el número 3 quedan borrados y reemplazados en el siguiente programa. La opción de DELETE debe ser especificada junto con la de MERGE.
- La sentencia de CHAIN no transfiere los tipos de variables, por lo que, los que se desea usar en los dos programas, han de ser especificados anteriormente.
- <número de línea 1> no recibe el efecto del RENUM (Ver 5.60).

Ejemplo:

CHAINA"PRGM-A" Se carga y se ejecuta el "PRGM-A", en vez del programa que se está ejecutando.

CHAINΔ"PRGM-A", 200, ALL "PRGM-A" es cargado en vez del programa que se está ejecutando. Todas las variables del programa precedente son transferidas al PRGM-A, y la ejecución del PRGM-A empieza a partir del número de línea 200.

CHAINCMERGEΔ"SUB-A", DELETE 1000-2000

Se saca la parte del programa actual, entre los números de línea 1000 y 2000 para intercalarlo al SUB-A. Se ha creado un nuevo programa. El control empieza un nuevo programa desde el principio.

5.5 CLEAR

Esta instrucción borra los datos de las variables.

Existe la función opcional para reinicializar la dirección máxima y la capacidad de la memoria de stack del programa BASIC.

```
CLEAR [<p1>][, <p2>]
```

- Cuando se ejecuta la sentencia de CLEAR, se le asigna 0 a todas las variables numéricas y una string nula a las variables de cadena.
- Cuando se especifica <p1>, puede reinicializarse la dirección máxima en un programa de BASIC. Como norma, la dirección máxima se especifica en el momento de la inicialización (ver 1.1) pero puede reinicializarse a través de esta sentencia.
- Cuando se especifica <p2>, la capacidad de la memoria de stack existente en el intérprete de BASIC será reinicializada a <p2> bytes. (Normalmente, en SBASIC, se reservan 512 bytes para la memoria de stack. Sin embargo, el tamaño de la área para la memoria puede volver a ser asignada mediante la especificación de <p2> para ampliar el anidado de bucles). La capacidad del stack debe ser de más de 256 bytes.

Puede usarse una expresión aritmética para escribir <p1> y <p2> .

No se puede especificar la sentencia de CLEAR en el bucle de FOR/NEXT, de WHILE/WEND o en las subrutinas.

Ejemplo:

- CLEAR
- CLEAR &HCFFF
- CLEAR &HCFFF, 1024
- CLEAR , 1024

5.6 CLOSE

La sentencia de CLOSE cierra el fichero del disco que se está usando.

```
CLOSE [[#]<número de fichero>]{[#]<número de fichero>}
```

- <número de fichero> indica el fichero actualmente abierto y se especifica con una expresión aritmética.

- Cuando se omite la especificación de <número de fichero>, todos los ficheros abiertos se cierran.
- La sentencia de CLOSE hace desaparecer la conexión entre un fichero de disco en particular y su número de fichero, usando el mismo número para abrir un nuevo fichero.
- El fichero de salidas secuenciales lleva a cabo una entrada de datos de la memoria intermedia (buffer) al fichero del disco mediante la ejecución de la sentencia de CLOSE.
- Cuando se ejecuta la sentencia de END (ver 5.15) o la de NEW (ver 5.45), todos los ficheros abiertos quedan cerrados de manera automática incluso en el caso que no se especifique la sentencia de CLOSE. La sentencia de STOP (ver 5.66) no sirve para la función de cierre.

Ejemplo:

- | | |
|--|--|
| <ul style="list-style-type: none"> • CLOSE • CLOSE#1, #5 | <p>Todos los ficheros quedan cerrados.</p> <p>Unicamente se cierran dos fiheros que estén abiertos con los números de fichero 1 y 5.</p> |
|--|--|

5.7 CLS

La sentencia CLS borra la pantalla.

CLS

- Al especificar la sentencia de CLS, las imágenes visuales quedan borradas. El Cursor aparece en la posición de la Línea 1 y la Fila 1 (la de la parte superior izquierda de la pantalla).

5.8 CONT

El CONT hace empezar de nuevo el programa que se había parado.

CONT[N]

- El programa que se había suspendido mediante el uso del STOP o a través de la tecla de <BREAK> queda reanudado.
- En el caso de que se hayan hecho modificaciones en el programa mientras estaba parado, el CONT no será válido.

Además de las funciones que se acaban de mencionar, el CONT N también posee las siguientes:

- a) Imposibilita la suspensión temporal de una ejecución de programa, mediante la tecla de BREAK.
- b) En el teclado existe la entrada
- c) La secuencia de escape puede ser confirmada en el bucle de INKEY\$.

Cuando se vuelve a entrar el CONT en el modo de CONT N, las funciones antes mencionadas de a), b) y c) quedan canceladas.

5.9 DEF FN

La sentencia de DEF FN define el tipo de funciones del usuario para que fuedan ser especificadas en una línea.

```
DEF FN<nombre> [(<parámetro> {,parámetro>} )]=<expresión>
```

- <nombre> sirve para llamar la función definida a través de la sentencia de DEF FN, basándose en la misma norma que en el nombre de la variable. (una cadena alfanumérica encabezada por una letra y los dos primeros caracteres identifican el tipo de que se trata.) El formato "FN<nombre>" sirve para la llamada.
- <parámetro> puede ser omitido. (Las funciones pueden ser definidas sin necesidad de parámetros.) Cuando no se omite, el parámetro queda asignado por las variables en <expresión>. Entonces, queda reemplazado por el parámetro (el parámetro especificado al llamar la función) para evaluar el valor de la expresión.
- <expresión> define la evaluación del método de función y queda incluida en una línea de sentencia DEF FN de hasta 255 dígitos.
- Las variables en la expresión simplemente hacen determinar el método de evaluación. No causa ningún efecto, incluso en el caso de que se usen las mismas variables que en el programa.
- Las variables en la expresión no se convierten, necesariamente, en los parámetros de la sentencia. Cuando se usa una variable en particular, puede ser considerada como un parámetro real en la calculación, y cuando no se usa, se tendrá que calcular el valor real de la variable.
- El valor de la función calculada se convierte en el código del tipo. Si la conversión no es correcta, aparece un error.
- La sentencia DEF FN deberá ser definida antes de la referencia.
- No se usa en Modo Directo.

- No deberá usarse ninguna función del usuario que no se haya definido, como un elemento de la expresión.

Ejemplo:

```
DEF FNA (X) = 3.14*2*X+B
B=5.0
C=FNA(25.0)           C se calcula mediante 3.14*2*25+5.
```

5.10 DEFINT/DEFSNG/DEFDBL/DEFSTR

Estas sentencias definen el tipo de variables con el primer carácter alfabético que también deberá usarse para los nombres de las variables.

$\text{DEF} \langle \text{tipo} \rangle \langle \text{caracter 1} \rangle [-\langle \text{caracter 2} \rangle \{, \langle \text{caracter 1} \rangle [-\langle \text{caracter 2} \rangle]\}]$

- Tanto INT, SNG, DBL como STR es asignado al $\langle \text{tipo} \rangle$.
- $\langle \text{caracter 1} \rangle$ es el primer carácter alfabético y $\langle \text{caracter 2} \rangle$ indica uno de los caracteres restantes que siguen el primero. El tipo del nombre de la variable que empieza con el alfabeto de $\langle \text{caracter 1} \rangle \langle \text{caracter 2} \rangle$ se escribe en la especificación de $\langle \text{tipo} \rangle$.
- Antes de esta declaración, se interpreta los códigos de especificación del tipo (ver punto (2) del 3.3). Con respecto a una variable con ningún código especificando el tipo, se atribuye el número real de precisión simple.
- La especificación del tipo a través de la declaración DEF puede volverse a definir en el programa.
- No existe ningún espacio entre DEF y $\langle \text{tipo} \rangle$.

Ejemplo:

DEFINT I-N	Las variables encabezadas por I,J,K,L,M o N serán de tipo entero.
DEFDBL X,Y,Z	Las variables encabezadas por X,Y o Z serán de tipo de número real de precisión simple.

5.11 DEF USR

La sentencia de DEF USR sirve para definir una subrutina escrita en un lenguaje Assembler. En la sentencia de CALL pueden escribirse varios parámetros pero por otro lado, la sentencia de DEF USR únicamente provee un parámetro y transfiere un resultado de ejecución simple, del programa de subrutina al programa principal. Esta es la diferencia más importante entre estas dos sentencias.

DEF USR[<n>] (parámetro formal)=(expresión)

- Unicamente se asigna un número entre 0 y 9 a <n>. Cuando éste se omite, quedará asignado el 0. En un programa BASIC, pueden definirse 10 tipos de funciones del usuario, mediante el empleo del lenguaje Assembler.
- <expresión> nos da la dirección de arranque de la subrutina.
- La expresión para la llamada de la dirección de arranque tiene el siguiente formato:

USR[<n>] (argumento presente)

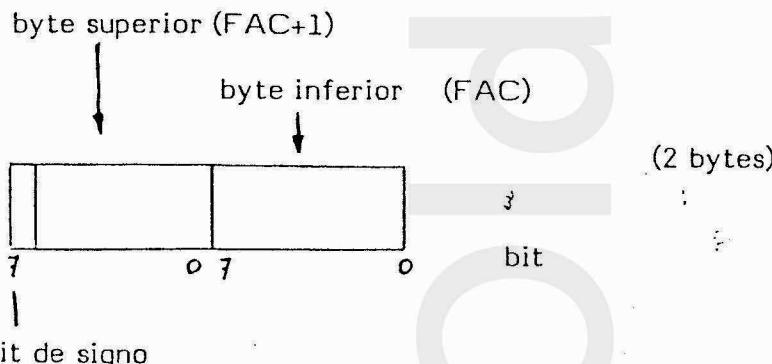
- <n>, en este caso, es el mismo número especificado en la sentencia de DEF USR.
- <parámetro> se refiere a uno de los elementos de datos (entero, número real, número real de precisión doble o string) que ha de ser transferido a la subrutina, de acuerdo con el siguiente proceso:

- 1) El código del tipo de datos es cargado en el Registro A.

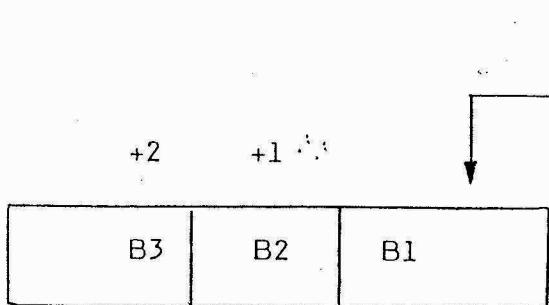
Tipo de parámetro	Código (Registro A)
Entero	2
Cadena de caracteres string	3
Número real de precisión simple	4
Número real de precisión doble	8

- 2) Cuando el parámetro es numérico, se carga en la Área de Función (FAC) de la memoria. La dirección de FAC es cargada en el registro HL. El contenido del FAC queda mostrado claramente de esta forma:

Entero (rA=2)



- 3) Cuando el argumento es un string, la dirección en memoria del inicio del string (de 3 bytes), se carga en el registro DE.



B1 : longitud de la cadena (número de caracteres entre 0 y 255)
 B2, B3 : dirección del encabezamiento de la cadena de caracteres (B2 es el byte inferior)

Una vez terminada esta preparación en el intérprete, el control salta a la dirección de arranque del programa de subrutina. El usuario deberá almacenar el resultado de la ejecución de la subrutina en el FAC para que de esta forma el control retroceda al programa principal. El intérprete transfiere al USRn los datos que han sido almacenados en el FAC. El resultado de ejecución han de ser del mismo tipo que el del parámetro llamado por la sentencia. Cuando la función es de tipo cadena, el resultado de ejecución deberá ser almacenada en la área en donde está la cadena del parámetro. La longitud de la cadena no variará. El índice de cadenas continua siendo el mismo.

Cuando la operación de pila se lleva a cabo en el programa de subrutina del usuario, la memoria de pila deberá ser restaurada en el estado original antes de que los datos sean restaurados. Si el puntero de pila cambia, el programa no procederá de forma normal. En el peor de los casos, el cambio del puntero de pila puede destruir los ficheros del disco.

Ejemplo:

A=&HA000

DEF USR5(p)=A+&H1000

X=USR5(Q)*2

5.12 DELETE

Esta palabra de instrucción borra una parte del programa en el Modo de Edición de Programas. Una vez ejecutada la instrucción, el control retrocede al modo de BASIC.

DELETE [*<número de línea 1>*] - [*<número de línea 2>*]

- Estas son las posibles combinaciones para la especificación de los números de línea:
 - (1) *<número de línea 1>-<número de línea 2>*
Borra la parte entre las dos líneas especificadas
 - (2) *<número de línea>-. (punto)*
Borra la parte desde el número de línea especificado, hasta el final del programa.
 - (3) *-<número de línea>*
Únicamente borra la línea del número que se ha especificado.
 - (4) *<número de línea>*
Únicamente borra la línea con el número de línea especificado.
- Cuando no existe programa en el rango determinado, aparece un error.
- La sentencia NEW (ver 5.45) se usa para borrar todo un programa entero.
- Un punto (.) indica la última línea del programa.

Ejemplo:

```
DELETE 500-1000  
DELETE 1200-.  
DELETE -790  
-DELETE 335
```

5.13 DIM

La sentencia DIM declara el tamaño de una tabla (array).

DIM *<variable>* (*<expresión>*, *<expresión>*), {*<variable>*
 (*<expresión>*, *<expresión>*)}

- <variable> indica el nombre del array (tabla).
- El valor de <expresión> determina el número de elementos del array en la dimensión correspondiente. El número de <expresión>, que sigue a la <variable>, indica el tamaño de la dimensión del array. En lo que respecta al tamaño, no existe ningún límite, así como tampoco en la cantidad de elementos, sin embargo, han de poderse incluir en la capacidad total del almacenaje de la máquina.
- La sentencia DIM se debe de declarar antes de utilizar el array correspondiente. Para obtener los valores evaluados para cada <expresión> se convierten en un número real, se redondean (contando en fracciones de 5 y despreciando el resto). Cuando el resultado es un valor negativo, se produce un error.
- Al dimensionar la tabla, todos los elementos se ponen a 0, en el caso de que la matriz sea numérica y si es de caracteres, serán sustituidos por cadenas de nulos.

El valor mínimo de un subscrito. Si se ejecuta DIM A(1Q), el número de elementos será de 11 valores, desde Q a 1Q. La sentencia de OPTION BASE (ver 5.49) puede modificar el valor mínimo hasta 1.

- Se deben especificar todas las matrices antes de usarse. Cuando el valor máximo del subscrito es de menos de 10, la matriz se podrá usar sin ser declarada. Cuando se usa una matriz por primera vez, se atribuirán las declaraciones de 1Q DIM, sin consideración a su tamaño.
- El dimensionado podrá ser cancelado mediante la sentencia ERASE (ver 5.16). (Cuando la capacidad total de la memoria está ocupada, el usuario podrá borrar las tablas innecesarias y declarar otras nuevas).

Ejemplo:

• DIM4A(10,20,10), B\$(20,3), C#(100)

• INPUT M,N
DIM A(2*M), B(5*N), C(M*,N)

A(3,5,8)=X+5

(Cuando se usa una tabla que tiene menos de 10 elementos, sin haberla dimensionado antes, queda atribuido A(10,10,10).)

DIM A(M,N) Cuando se especifica y en el caso de que no se haya asignado ningún valor a M y N, quedará atribuido A(0,0).

5.14 EDIT

EDIT es el comando para alterar (editar) las líneas.

EDIT <línea>

- Cuando se entra el comando de edición, se muestra la línea especificada, en la pantalla y el cursor muestra un destello en el comienzo de la línea siguiente. Edita (cambia, duplica, etc.) la línea al usar los siguientes comandos:

→
DEL
CR
BREAK

Desplaza el cursor, junto con el contenido de la línea.
Borra el carácter del lugar marcado por el cursor □.
Hace aparecer el resto de la línea y finaliza la edición.
Cancela la operación de edición y retrocede al modo READY.

5.15 END

END

- La sentencia concluye la ejecución del programa, cierra todos los ficheros abiertos y hace retroceder el control al Modo de BASIC.
- El usuario puede escribir la sentencia END en cualquier punto del programa. En el caso de que haya algún fichero abierto en el momento que se especifica la sentencia de END, este se cerrará. Luego, suspende la operación del programa y retrocede al Modo BASIC.
- Cuando no hay ninguna sentencia de END al final del programa, el sistema desarrolla unas operaciones similares, de forma automática (cerrando todos los ficheros y volviendo al Modo BASIC).

5.16 ERASE

El comando de ERASE, elimina las tablas deseadas, haciendo que el área correspondiente en la memoria pueda ser usada para otro fin.

```
ERASE <nombre de matriz>{,<nombre de matriz>}
```

- Al borrarse la matriz, la area correspondiente en la memoria, queda en blanco. Puede dimensionarse una nueva tabla usando el mismo nombre de la eliminada u otro cualquiera.
- El comando ERASE no puede usarse dentro de los bucles FOR/NEXT, WHILE/WEND ni en las subrutinas.

Ejemplo

```
10 DIM A(10,10), B(5,10), AS(100)
.
.
.
50 ERASE A,B
.
.
70 DIM A(50), C(10,5)
.
```

5.17 ERROR

El ERROR tiene dos funciones: una es la simulación de error para crear, de forma intencionada, los errores definidos en SBASIC y la otra es la función definida por el usuario para especificar los errores según su propia voluntad.

```
ERROR <expresión aritmética>
```

- Cuando el valor de la evaluación de la <expresión aritmética> concuerda con el número del error ya definido en SBASIC, aparece el correspondiente error. (ver lista de mensajes de error de SBASIC en el Apéndice B.)
- Cuando el valor es mayor que el número de error según la definición de SBASIC, servirá para la función definidora de error del usuario. El número de error definido por el usuario puede usarse mediante la ejecución de la subrutina de atrapar errores.
- El valor de <expresión aritmética> debe de ser un entero en el rango de 1 a 255. Cuando se sale de estos trámites, se produce un error. Cuando se convierte en un número real dentro del rango, se redondea (en fracciones de 5 y despreciando el resto).

- Cuando se entra un mensaje de error con un número inexistente en la lista, se produce un error.
- Cuando se produce un error, incluso en el caso de que se haya creado de manera intencionada, el sistema hace aparecer el mensaje correspondiente, suspende la ejecución del programa y retrocede al Modo BASIC.
- Cuando aparece un error, la sentencia ON ERROR COTO (5,46) sirve para:
 - detener el control retrocediendo al Modo BASIC;
 - llevar a cabo el procesamiento del error; y
 - generar la subrutina de atrapar errores con objeto de resumir el programa.

Ejemplo:

```
(1) ERROR 10
(2) .
.
110 ON ERROR GOTO 500
.
.
.
150 IF A>100 THEN ERROR 200
.
.
.
490 END
500 IF ERR =200 THEN 600
.
.
.
600 (cuando A>100)
650 RESUME NEXT
```

5.18 FIELD#

El FIELD asigna un nombre de campo, así como su longitud en el área de buffer para ficheros directos.

```
FIELD [#]<número de fichero>,<tamaño del campo> AS<variable string>
          {<tamaño del campo> AS<variable de carácter>}
```

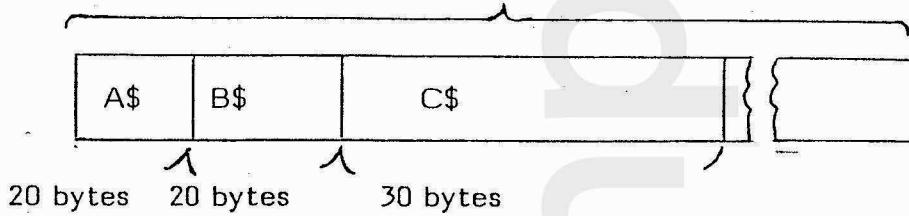
- <número de fichero> especifica el fichero en particular que se ha abierto. Se escribe con <expresión aritmética> .
- <tamaño del campo> indica la longitud (número de bytes) que han de ser asignados a la <variable de cadena>.

Ejemplo:

```
FIELD#1, 20 AS A$, 10 AS B$, 30 AS C$
```

Area de buffer

Area de buffer



- La sentencia FIELD, tal y como se ha mostrado , asigna un nombre a cada campo y determina su longitud. La sentencia LSET o RSET (ver 5.41) debe seguir a la sentencia de FILES, con objeto de hacer la entrada de los datos del fichero del disco en el buffer y la sentencia de PUT (ver 5.56) desarrolla la operación de escritura en el fichero del disco. El capítulo 8 describe estos procesos de forma detallada.
- La longitud total de los campos, especificado en la sentencia FIELD, no debe sobrepasar la longitud de un fichero aleatorio definido por la sentencia OPEN (ver 5.48)
- Cuando el usuario desea modificar el nombre de la longitud de los campos de una memoria intermedia:
 - (1) Reinicializa el programa mediante lo siguiente:
FIELD#1,0 AS A\$, 0 AS B\$, 0 AS C\$
 - (2) Entra otra sentencia FIELD con objeto de otra asignación.La variable de cadena definida en la sentencia FIELD no deberá volverse a definir en LAS SENTENCIAS INPUT, LET, READ, SWAP o MID\$.
- Para los datos de una variable, se puede especificar una variable simple. (No se podrá usar elementos de una tabla).

5.19 FILES/LFILES

[L] FILES [<nombre de fichero>]

- El sistema manda el nombre de fichero almacenado en el disco a la pantalla o a la impresora. FILES es para la pantalla y LFILES para la impresora.
- Cuando se omite el <nombre de fichero>, aparecerán todos los nombres de fichero del disco que, en aquel momento, se hallen en el Drive A.
- Cuando se incluye un punto de interrogación (?) en el nombre de fichero, se convertirá en un substituto para cualquiera de los caracteres. Por lo tanto, el sistema hará aparecer por la pantalla o bien imprimirá todos los caracteres posibles diferentes de ?.
- El capítulo 8 trata el tema de los ficheros con detalle.

Ejemplo:

FILES	Indica todos los ficheros en la Unidad A
FILES A "B:*./*"	Indica todos los ficheros en la Unidad B.
FILES A "*.*.BAS"	Indica todos los ficheros que tienen la extensión BAS.
FILES A "A?????.**"	Indica todos los ficheros cuyo nombre sea de 5 dígitos, empezando por A.

5.20 FOR/NEXT

Instruye al sistema para que vaya repitiendo lo que se describe entre el FOR y el NEXT. Las sentencias de FOR y NEXT deben usarse juntas.

FOR <variable> = <expresión aritmética 1> TO <expresión aritmética 2>
[STEP <expresión aritmética 3>]

NEXT [<variable> { ,<variable> }]

- <variable> indica el número de repeticiones que se han de llevar a cabo.
- <expresión aritmética 1> calcula el valor inicial para contar el número de repeticiones. <expresión aritmética 2> nos da el valor final. (El contador en la consola muestra, al operario, el número contado.) Una vez se han hecho las ejecuciones de las sentencias posteriores a la sentencia de FOR pero antes de la de NEXT, el valor del contador será aumentado mediante el valor de <expresión aritmética 3>. Cuando el valor del contador no alcanza el valor final, el control retrocede a la sentencia siguiente a la de FOR y va repitiendo así, hasta que el valor del contador alcanza el valor final.

Al alcanzar el valor final, el control salta a la sentencia siguiente a la de NEXT.

- En el caso que el valor de la inicialización del contador sobrepase el valor final, no será necesario la ejecución del bucle. El control salta a la sentencia escrita junto a la de NEXT. Cuando se describen varias sentencias NEXT para una sola sentencia FOR, el control salta a la sentencia siguiente a la de NEXT con el número de línea menos.
 - Cuando se omite la declaración de STEP, queda asignado el valor "1" para la <expresión aritmética 3> .
 - El bucle de FOR y NEXT pueden incluirse. Cuando se hace la inclusión de datos, se deben definir las variables para el valor del contador, en cada uno de los bucles.
 - En el caso de que el valor final sea el mismo que el de los bucles incluidos, las variables se podrán escribir en una sentencia NEXT. Sin embargo, las variables del contador para el bucle interno deberán escribirse en la parte izquierda de la variable del bucles exterior.
 - Cuando se omiten las variables, en una declaración de NEXT, se les atribuye las mismas variables que en la anterior sentencia de FOR.
 - No se puede asignar un número de precisión real a <variable> (la variable de control de repetición en un bucle).

Ejemplo:

```

(1) FOR 1% = 1 TO 10
    .
    NEXT
(2) FOR A=1 TO 10^4 STEP A 0.5
    FOR B=1 TO 20
    FOR C=1 TO - 10^4 STEP A -1
    .
    .
    NEXT C,B,A
(3) FOR X=3E-7 TO 10E-8STEP-1E-7
    .
    .
    NEXT X
(4) INPUT M,N
    .
    FOR I=M TO M+N
    .
    NEXT

```

5.21 GET

El GET carga los datos del fichero de acceso aleatorio en el buffer definido mediante la sentencia FIELD (ver 5.18).

```
GET [#] <número de fichero> [, <número de registro>]
```

- <número de fichero> es el número bajo el cual se ha abierto un fichero en modo "R".
- <número de registro> es el número de registro del fichero, en un rango entre 1 y 32767. Cuando éste se omite, se le atribuye el mismo que el usado en el registro siguiente, del último GET o PUT.
- Tanto <número de fichero> como <número de registro>, pueden escribirse en expresión aritmética .
- En lo concerniente al uso de los ficheros de discos, consulte el Capítulo 8 de este manual.
- Si un número de registro sin ser PUT es GET, el contenido del registro será el del que no se define.

— Ejemplo:

- GET # 1
- GET # 2, 501
- GET # N, REC%

5.22 GOSUB/RETURN

GOSUB o RETURN instruye al programa para bifurcarse a una subrutina o retornar de la misma al programa.

```
GOSUB <número de línea>
      ...
      RETURN
```

- <número de línea> es el número de la primera línea del programa de la subrutina. <número de línea> deberá ser constante entera .
- La sentencia de RETURN se hace al final del programa de subrutina. Cuando se ejecuta el RETURN, el control retrocede a la siguiente sentencia a la de GOSUB.
- Pueden incluirse llamadas a la subrutina. (Dentro de una subrutina se puede invocar a otra). Sin embargo, una subrutina no puede invocarse a sí misma.

Una vez ejecutadas las sentencias de GOTO, STOP o END, se podrá entrar un programa de subrutina en cualquier punto del programa principal. Si no se procede de esta forma, el control principal se desplaza al programa de subrutina.

Ejemplo:

```
300 GOSUB 500          200 GOSUB 1000
.
.
.
490 GOTO 600          999 END
500 .                  1000 .
.
.
590 RETURN }           } Subrutina
600 .                  1200 RETURN
.
.
.
999 END
```

5.23 GOTO

Este es una instrucción de bifurcación incondicional.

GOTO<número de línea>

- <número de línea> es la constante entera que indica el destino (número de línea para la bifurcación)
- Cuando se ejecuta la sentencia de GOTO, el control va hasta el número de línea especificado.
- Si no se encuentra el número de línea especificado en el programa, aparece un error.
- GOTO también se describe como "GO TO".

Ejemplo:

```
200 GOTO 500
.
.
.
500 A=A+1
.
.
```

5.24 IF/THEN/ELSE (SI/ENTONCES/SINO)

La sentencia de IF es una instrucción divergente condicional por la cual, en ciertas circunstancias, se pueden efectuar distintas funciones.

```
IF <expresión> THEN <sentencia> { : <sentencia>} | <número de línea>
  [ ELSE <sentencia> { : <sentencia>} | <número de línea> ]
```

o

```
IF <expresión> GOTO <número de línea>
  [ ELSE <sentencia> { : <sentencia>} | <número de línea> ]
```

- <expresión> indica la condición para la divergencia. Cuando el valor de evaluación de <expresión> no es 0, el control ejecuta la sentencia, después de la sentencia de THEN o de GOTO. Cuando es 0, se ejecuta la cláusula ELSE. En caso de que se omita la cláusula ELSE, el control pasa a la sentencia siguiente a la de IF, una vez se haya ejecutado la cláusula de THEN.
- La sentencia IF se puede escribir siguiendo a un THEN o ELSE (Esto significa que la sentencia IF puede ser anidada). No existe ningún límite en cuanto a la profundidad de una inclusión pero se ha de tener en cuenta que toda la sentencia no exceda los 255 caracteres.)
- Cuando se incluyen las sentencias IF, queda entendido que la sentencia ELSE se incluirá en la sentencia IF anterior. La sentencia ELSE no podrá excluirse de un bucle de las sentencias IF. Se puede omitir solamente en la sentencia IF más externa.

Ejemplo:

```
IF A>3 THEN A=A-1:C=D:H=H+2:GOTO 200
      ELSE 300
IF A=B GOTO 500 ELSE A=0: GOTO 600
```

```
IF A=B THEN IF C>D THEN 200
      ELSE 300
      ELSE IF C<D THEN 400
      ELSE 500
```

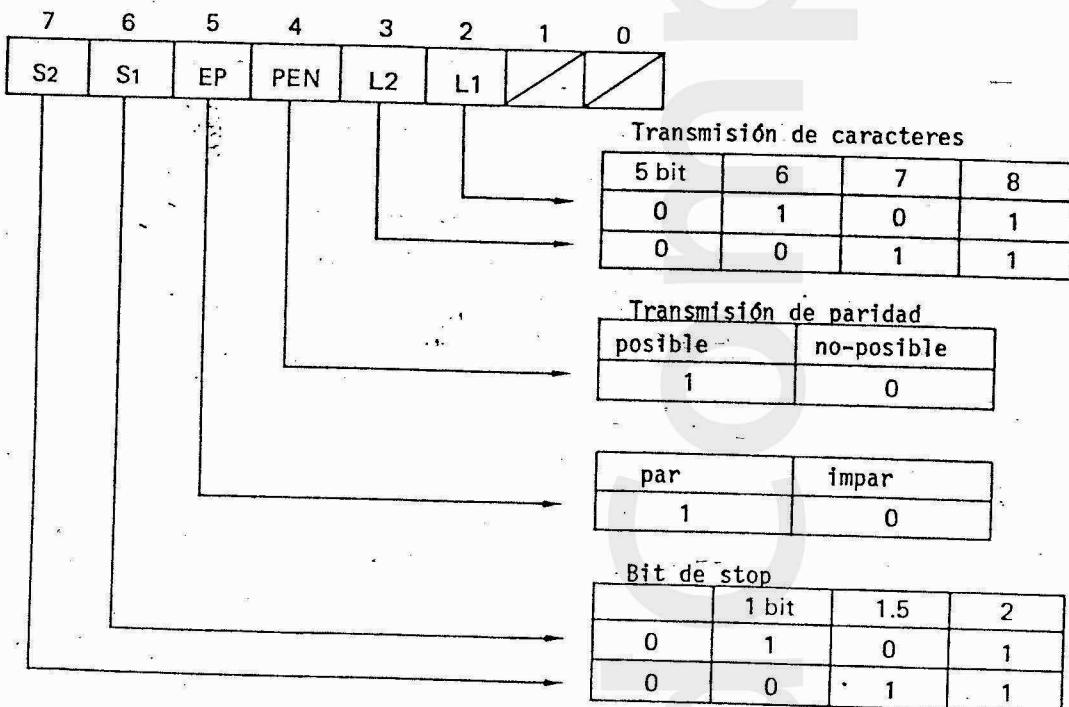
IF A\$ = "ZZZ" GOTO 100 } indiferentes
IF A\$ = "ZZZ" THEN 100
IF A\$ = "ZZZ" THEN END — valido
IF A\$ = "ZZZ" END — No valido

5.25 INIT%

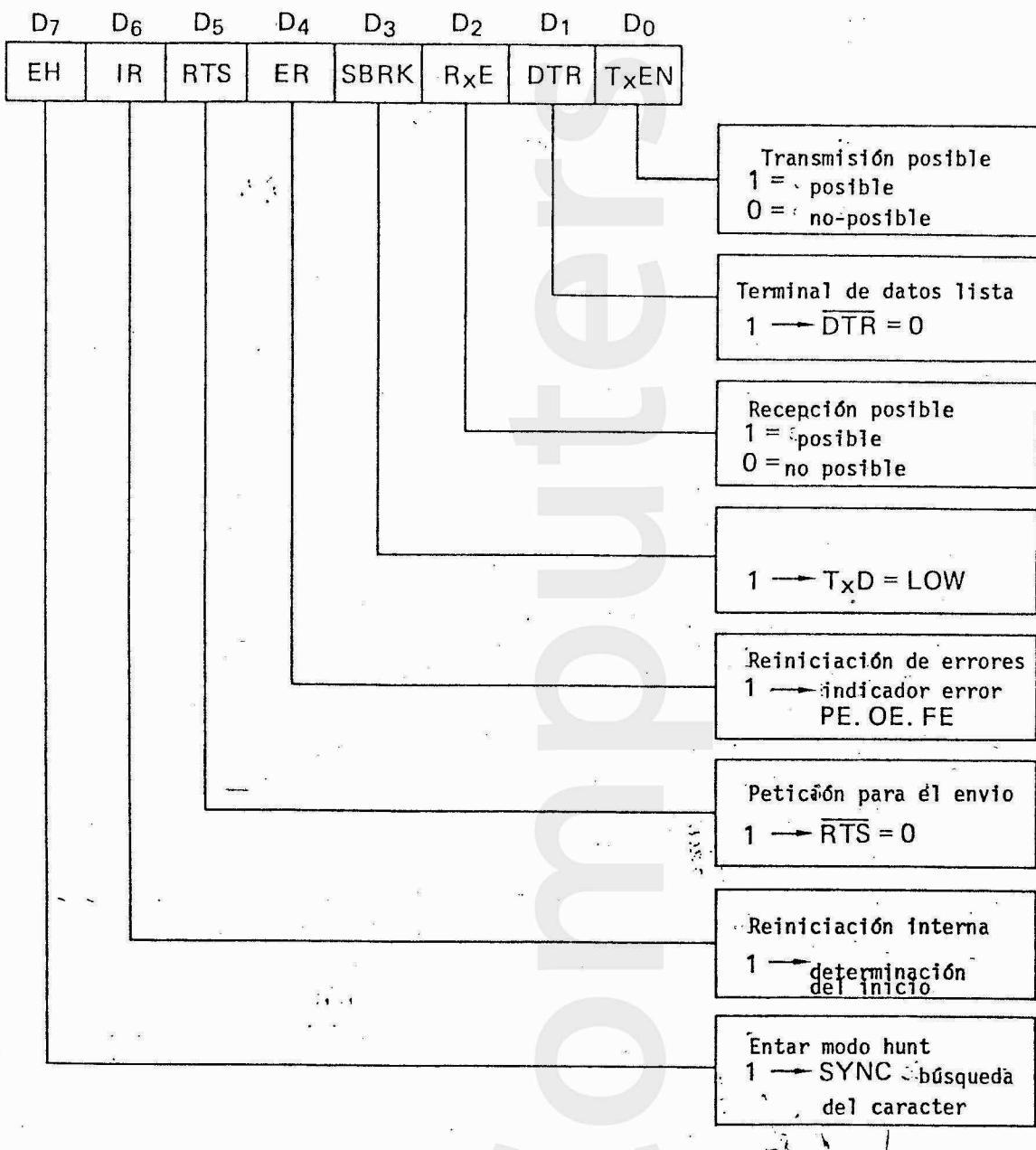
La sentencia de INIT% establece los parámetros de transmisión de mandar/recibir

INIT%<número de > , <expresión 1>, <expresión 2>

- Al <número de canal> se le asigna 1 o bien 2. LINE-1 se usa para Tarjeta de comunicaciones (FS-203) de I/F, y la LINE-2 para alguna posible expansión del sistema.
- <expresión 1> establece el parámetro de 1 byte para especificar un modo de mandar/recibir.
- <expresión 2> establece el parámetro de 1-byte para especificar un comando mandar/recibir.
- <expresión 1> = fijación de modo (1 byte)



< expresión 2>= selección del comando (1 byte)



- Si no se ejecuta la sentencia INIT%, el sistema hace automáticamente la siguiente inicialización.
Caracteres: 8 bits, sin paridad.
2 bits de stop.Bit de stop: 2
 - La selección del rango de baud debe llevarse a cabo mediante la tarjeta I/F (FS- 203 JP2, Consulte la guía del usuario del RS- 232C.)

Nº	8	7	6	5	4	3	2	1
Rango de banda		9,600	4,800	2,400	1,200	600	300	150

- Seleccione el JP1 del FS-203 a 1.
 - Seleccione el JP3 del FS-203 a 2 y 4.

5.26 INPUT

después CR sigue en la misma linea
Espera Valor con ? Espera Valor con ?
Esta instrucción tiene como función la entrada de datos mediante el uso del teclado.

INPUT [;][< constante de carácter>;] <variable> {,<variable>} ①
INPUT (<expresión aritmética>)[;] [< constante de carácter>;]<variable> ②
(de 1 a 255) "XXXXX" 1337

Solo una Variable

- Cuando el control llega a la sentencia de INPUT, el sistema suspende el procesamiento del programa y espera la entrada de datos a través del teclado. Una vez entrados los datos, con <CR> al final, queda almacenado como variable y el programa pasa al siguiente paso.
- El número de elementos de datos que han de ser entrados es el mismo que el de las variables usadas en la sentencia de INPUT. Cuando se entran los datos, se debe usar el delimitador (, =coma) entre cada elemento de datos.
- < constante de carácter> aparece en la pantalla mientras el sistema está esperando la entrada de datos. Este mensaje informa, al operador, de los datos pedidos. Después de < constante de carácter>, el usuario deberá introducir bien un punto y coma (;) o una coma (,). En el caso que se seleccione el punto y coma, aparecerá por la pantalla la <constante de carácter> junto con un punto de interrogación y si simplemente se usa una coma, no seguirá nada a la constante de carácter.
- Si se omite < constante de carácter>, en la pantalla sólo aparecerá un punto de interrogación (?). En el caso de que existan más de dos variables, en la sentencia de INPUT, el sistema hará aparecer dos interrogantes (??), hasta que se entren los datos para la segunda sentencia y se reciban las últimas variables.
- El punto y coma (;) entre INPUT y < constante de carácter> es el control del código <CR>, después de la entrada de datos. Cuando se escribe punto y coma, el Cursor permanece junto con el elemento de datos que se ha escrito, ya que <CR> no aparece por la pantalla. Se verá otro elemento de datos después del anterior.

Cuando se omite el punto y coma, aparecerá por la pantalla <CR> para indicar que la entrada de datos ha terminado y el Cursor retrocede y avanza hasta la siguiente línea.

- El formato ① de INPUT no controla el dígito de los datos de entrada pero permite varias variables. El formato ② de INPUT controla el dígito de los datos de entrada pero únicamente acepta una variable.
- En el caso del ①, se debe entar el mismo número de datos que para las variables, separando los datos mediante una coma. Si la clase da datos entrados no correspondiera a los que estamos definiendo, los datos se transforman para coincidir con el tipo de variable, almacenándolo posteriormente. Si la clase de transformación no es posible, "Bad data" aparece en la pantalla y el control retrocede al comando del modo input. En el caso de que un dígito de los datos sobrepasase un dígito de una variable, la parte excediente no sería tomada en cuenta (no aparecería por la pantalla).
- En el caso de ②, se controla el dígito usado para los datos. Debe ser dentro de un dígito de <expresión aritmética> de valor (1-255). Si se excediese del límite, la parte sobrante quedará ignorada (no visualizada) y únicamente se acepta <CR>.
- En el caso de que el tipo de variable usada no fuera idéntico al usado para los datos, "Type mismatch error" aparecerá en la pantalla. Una coma se considera como fin del procesamiento de datos.

- Cuando el tipo de la <variable> y el de los datos no son iguales, el contenido de los datos deberá ser transformado con objeto de que coincida con el de la variable. Si no fuera posible esta transformación, el sistema hará aparecer por la pantalla el mensaje de "Bad data" y quedará a la espera de otra entrada de datos. Cuando el número de elementos de datos entrados exceda al de las variables en la sentencia de INPUT, los datos que sobrepasen tal número no serán tenidos en cuenta.
- En lo que respecta a la entrada de datos de caracteres, la cadena de caracteres que debe entrarse no podrá incluir ninguna coma (,). Una coma es el delimitador para separar los elementos de datos. Para la entrada de una cadena de caracteres en la cual se tenga que incluir la coma, se usará la sentencia de LINE INPUT (ver 5.34).

Ejemplo:

- INPUT A, B, XS
- INPUT "INPUT DATA", A
- INPUT (15)X

INPUT (3); "XXXX", X

INPUT; "XXXX", X, Y, Z

INPUT (3); "XXXX"; X

INPUT; "XXXX"; X, Y, Z

5.27 INPUT#

INPUT# lee datos de un fichero secuencial y los asigna a las variables especificadas.

INPUT# <número de fichero>, <variable>{ ,<variable>}

- <número de fichero> es el número mediante el cual queda abierto un fichero en el modo "I".
- <variable> almacena los datos surgidos del fichero. El tipo de variable debe coincidir con el de los datos.
- El sistema lee los datos saltándose el espacio y el <LF> del principio.
- Cuando los datos de carácter empiezan con el signo de comillas (""), los datos se leen directamente como un solo elemento, hasta que salga el siguiente signo de comillas.
- Si los datos de carácter comienzan de otra forma, un elemento de datos continua hasta que aparece una coma (,), CR, LF o el carácter 255. En el caso que el usuario prefiera usar coma (s) en el elemento de datos, existe la sentencia de LINE INPUT# (ver 5.35).

Ejemplo:

- OPEN "I" #3, "FILE -A3"
- INPUT# 3, A\$, B\$

5.28 INPUT%

INPUT% lee directamente los datos del port de comunicaciones de la unidad RS-232C, almacenándolo en la variable.

```
INPUT% <número de port>, <variable de carácter>,  
           ,<variable de carácter>
```

- <número de port> puede ser 1 o bien q coincidiendo con el conector de LINE-1 o LINE-2 del
- <variable de carácter> almacena los datos entrados del port. Todos los datos transmitidos son procesados como datos de carácter.
- Una coma (,) sirve para separar los datos entrados. Los datos que siguen a una coma serán almacenados en la siguiente área para la variable. Para la entrada de datos en los que se incluya una coma, podrá usarse la sentencia de LINE INPUT%.

Ejemplo:

```
INPUT% 1, &HFF, 3  
.  
INPUT% 1, A$, B$, C$  
.  
.
```

5.29 KEY

El Key asigna cadenas de caracteres a cada una de las teclas de función (PF1 a PF5). Cuando se hayan asignado varias cadenas de caracteres, con anterioridad, el sistema desarrolla las mismas operaciones que en las cadenas de caracteres, mediante las teclas necesarias de PF.

```
KEY <PF número de tecla>, <expresión de cadena>
```

- <número de tecla PF> indica el número de una tecla PF para la cual se debe asignar una cadena. Se escribe con <expresión aritmética>. El número de tecla especificado deberá ser del 1 al 5.

- <expresión de cadena> consiste de caracteres de hasta 15 dígitos y es asignada a las respectivas teclas de función. La parte de la cadena que sobrepase este límite no será tomada en cuenta.
- En el proceso de la operación del número de tecla especificado, la asignación no será válida si su contenido se modifica para el mismo número de tecla.

Ejemplo:

CHR\$ (13)

KEY 5, "RUN" +*CHR\$(&H0D)*

La función RUN<CR> es asignada a PF5.

KEY 1, "TAB(";

KEY 2, "FLD\$ ("+*CHR\$ (&H22)+"N" +CHR\$ (&H22)?")"*)"

La función FLD\$ ("N") es asignada a PF2.

5.30 KEY LIST/KEY LLIST

La sentencia KEY LIST sirve para listar el contenido de los datos asignado a las teclas de función, debido a que el sistema requiere la visualización de la lista por la pantalla. La sentencia KEY LLIST manda la lista a la impresora.

KEY LLIST

- Al hacer la impresión mediante la sentencia KEY LLIST (KEY LIST), los códigos de control se convierten en interrogantes (?).

5.31 KEY LOAD/KEY SAVE

La sentencia KEY SAVE tiene como función el almacenar la lista de asignaciones en el fichero del disco y la de KEY LOAD pasa la lista de asignaciones a la memoria.

KEY LOAD <nombre de fichero>
KEY SAVE <nombre de fichero>

- nombre de fichero es el nombre del fichero en el que se ha de salvar la lista de asignaciones de las teclas de función. En el disco, se pueden almacenar varios tipos de secuencias de asignaciones y el tipo de lista más adecuado dependerá del contenido de la operación en particular.

Ejemplo:

KEY SAVE "B:K-TBL-1"

KEY LOAD "B:K-TBL-1"

5.32 KILL

Esta instrucción borra un fichero del disco.

KILL <nombre del fichero> {, <nombre del fichero> }

nombre del fichero especifica el fichero para ser borrado. El fichero abierto no puede especificarse.

Cada carácter puede ser sustituido por un interrogante (?) especificando el nombre del fichero. Un asterisco (*) también puede ser asignado a todos los nombres del fichero o nombres del fichero extendido.

La instrucción KILL funciona con todos los ficheros del disco.

Ejemplo:

KILL "PRGM.BAS", "FILEA.DAT", "FILE.ABC"

KILL "TEST? ?.*"

KILL "B:*.**"

KILL "C:*.DAT"

5.33 LET

La sentencia LET introduce el almacenamiento de datos dentro del área de las variables.

[LET] <variable> = <expression>

La palabra LET puede ser omitida.

Cuando se escribe variable con un valor numérico y el valor de la evaluación real de la expresión es alfanumérica, o viceversa, entonces ocurre un error.

- o Cuando <variable> y <expression> son ambas numéricas pero sus tipos son discrepantes, la operación de <expression> se ejecuta primero, entonces la conversión del valor de la evaluación dentro del código del tipo viene a continuación. En el caso de que la ejecución no pueda llevarse a cabo, se produce error.

Ejemplo:

```
A=1: B=2: C=3  
A=A+1  
A$="ABC"+CHR$(&H41)  
A =0.123456789D-5  
A%=231.5
```

Entrada y resultado
(nada) Entrada y resultado

5.34 LINE INPUT

La sentencia LINE INPUT sirve para almacenar las cadenas de caracteres entrados en la variable alfanumérica.

```
LINE INPUT[; <constante de cadena>]; <variable alfanumérica>
```

Igual que INPUT, el LINE INPUT lee los datos de una cadena introducidos en una variable alfanumérica, <CR> deberá teclearse al final de los datos.

constante de cadena es un mensaje que el operador visualiza al momento de teclear. No aparece ningún signo de interrogante (?).

Un punto y coma (;) se colocará entre la palabra (LINE INPUT) y <constante de cadena> para el control del código <CR> que se ejecuta cuando se deja de teclear. Cuando existe el punto y coma, el sistema no envía ningún código <CR> a la pantalla. El Cursor aparece inmediátamente después de entrar los datos. El control del sistema va al siguiente paso del programa en el cual el Cursor aparece inmediátamente después de haber entrado los datos. Cuando no existe el punto y coma, el código <CR> sale a la pantalla para indicar que se ha dejado de teclear y el Cursor se dirige al principio de la siguiente linea (carro de retorno y avance de linea).

Cuando el INPUT está especificado, una coma (,) actúa como un delimitador de datos, sin embargo, en el caso del LINE INPUT, la coma (,) también se lee como uno de los datos.

- <expresión aritmética> revisa el número de caracteres de los datos entrados. <Las expresiones aritméticas> con los caracteres del 1 al 255 pueden ser entradas. Si las expresiones aritméticas exceden de 255, sólo se acepta el <CR>. Las <expresiones aritméticas> decimales son redondeadas automáticamente, pero si se exceden del límite, aparecerá un mensaje de error.

Ejemplo:

LINE INPUT	"address";AD\$
LINE INPUT;	"data key-in";X\$
LINE INPUT	(25);"address";A\$
LINE INPUT	(A+B) Y\$

5.35 LINE INPUT

La sentencia LINE INPUT# tiene el mismo formato que LINE INPUT y lee los datos del fichero (fichero secuencial) en vez de desde el teclado.

LINE INPUT #<número del fichero>, <variable de cadena>

- <número del fichero> es el número bajo el cual se abre el fichero en modo "I".
- Un elemento de los datos que haya que leerse fuera del fichero secuencial dentro de una <variable de cadena> continua desde el principio hasta <CR>. (En la especificación INPUT#, una coma (,) es como un delimitador de datos, sin embargo, en la especificación LINE INPUT, los datos se leen hasta que aparece <CR> incluyendo la(s) coma(s). Ver 5.26)

Ejemplo:

OPEN "I", 2,"FICHERO DE DATOS"

LINE INPUT 2.X\$

.

5.36 LINE INPUT%

El LINE INPUT% tiene el mismo formato que el LINE INPUT y lee los datos provenientes del port de comunicaciones RS-232C introduciéndolos en la variable.

LINE INPUT%<número del port>, <variable alfanumérica>

- o Tanto el 1 como el 2 se le asigna al <número del port>. Normalmente se usa el 1, y el 2 es para posibles futuras expansiones del sistema.
- o Los datos son leídos del port en serie introducidos en la <variable de cadena>, aunque se incluya una coma (,). La operación de lectura termina cuando el <CR> aparece en la cadena.

Ejemplo:

```
LINE INPUT%2, A$
```

5.37 LIST/LLIST

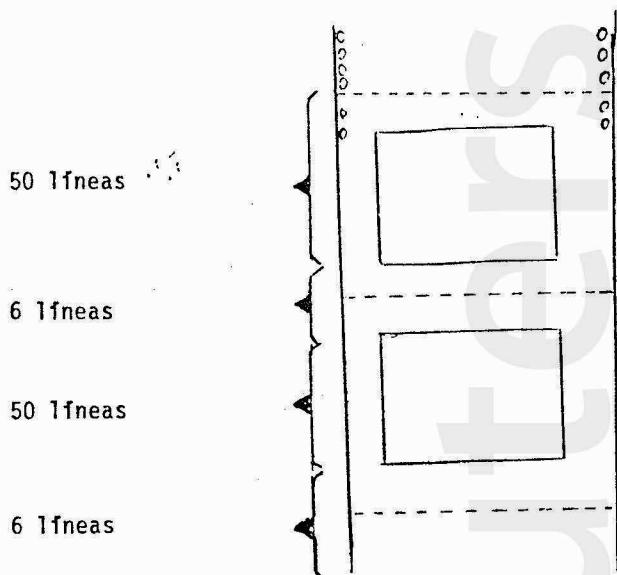
La sentencia LIST lista en la pantalla el programa almacenado en la memoria. El programa sale por pantalla a través de la sentencia LIST o por la impresora a través de la sentencia LLIST.

LIST[<número de línea 1>]-[<número de línea 2>]	
LLIST[<número de línea 1>]-[<número de línea 2>], [constante del entero]	

- o La especificación del <número de línea> está interpretada de la siguiente manera:
 - (1) No especificado Visualiza un listado.
 - (2)<número de línea 1>-<número de línea 2> Lista desde el número de <línea 1> hasta el número de <línea 2>
 - (3)<número de línea>- Lista desde la línea especificada con el número de línea hasta el final del programa.
 - (4)<número de línea># Lista desde el comienzo del programa hasta la línea que está especificada con el número de línea.
 - (5)<número de línea> Sólo lista los datos de la línea especificada. Un punto (.) indica el final de la línea del programa.
 - (6) LIST Lista los datos de la última línea del programa.
- o El <número de línea> sólo se escribe con una <constante de enteros>.
- o Cuando la <constante enteros> está escrita en la sentencia LLIST, entonces una página consiste del número (entero) de líneas. Aparecen seis líneas en blanco entre las páginas.

Ejemplo:

LLIST,50→



La constante que sirve para especificar el número de líneas para una página debe estar en el rango entre 0 y 255. Cuando es 0, las líneas aparecen sin ningún alineamiento de página. (Produce el mismo efecto que tiene la sentencia sin ninguna especificación.)

- Cuando se omite la <constante del entero>, las 60 líneas con las 6 líneas en blanco quedan alineadas en una página. (Se tiene en cuenta las 11 pulgadas de los márgenes de las hojas). Si no se realiza la opción de la <constante de entero>, entonces las líneas salen sin ningún alineamiento de página, y no se produce ningún espacio en blanco entre las páginas.
- El listado se para pulsando la barra de espacio y vuelve a continuar pulsando cualquier tecla. Para más información vea el capítulo 7.
- Cuando se imprime un programa, todos los caracteres en las columnas 8,9,E,F de la tabla de códigos, se representan con el signo "?".

Ejemplo:

LIST

Lista en la pantalla el programa entero.

LIST 100-500

Lista en la pantalla los datos que van desde la Línea 100 a la 500.

LIST -250

Lista los programas desde el comienzo hasta la Línea 250.

LLIST 100-1000,

Imprime, con 60 líneas por página, los datos que van desde la Línea 100 a la 1000.

LLIST 1000-,50

Imprime, con 50 líneas por página, los datos que van desde la Línea 1000 hasta el final.

LLIST

Imprime los programas enteros por alineamiento de página pero sin ningún espacio en blanco.

LIST.

Sólo visualiza la última línea.

5.38 LOAD

LOAD carga en memoria un programa que está en el disco.

LOAD <nombre del fichero> [R]

- o <El nombre del fichero> especifica el fichero del disco en el cual ha sido salvado el programa.
- o Cuando se ejecuta la sentencia LOAD, el sistema cierra todos los ficheros abiertos y destruye el actual programa en memoria, y carga el programa con el <nombre del fichero>. Después de la ejecución, el control vuelve al modo BASIC. Cuando se incluye ",R", el sistema conserva los ficheros abiertos, lee el programa desde el disco, e inmediatamente ejecuta el programa entero.
- o Cuando no existe ningún fichero en el disco, que esté especificado por el <nombre del fichero>, entonces el actual programa permanece inalterado en la memoria.

Ejemplo:

```
LOAD    "PRGM-A"  
LOAD    "PRGM-A",R
```

5.39 LOCATE

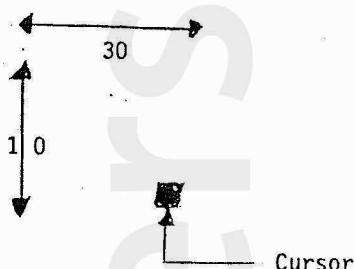
LOCATE envía el Cursor a la posición que se desee en la pantalla.

LOCATE <expresión 1>, <expresión 2>

- o <La expresión 1> y la <expresión 2> constituye la coordenada para mostrar la posición del Cursor. La coordenada consiste en posiciones de 25 líneas horizontalmente y de 80 caracteres verticalmente. La posición de un Cursor puede especificarse con este rango.
- o <La expresión 1> indica la posición horizontal (puntero de línea) en el rango entre 1 y 80. El 1 indica el final a la izquierda y el 80 a la derecha.
- o <La expresión 2> indica la posición vertical (puntero de línea) en el rango entre 1 y 25. El 1 indica el tope de arriba y el 25 el final.
Cuando el valor evaluado de expresión incluye una parte decimal, éste queda redondeado automáticamente (contando en fracciones sobre 5 como unidad, despreciando el resto). Si el valor se excede del rango, entonces aparece un error.

Ejemplo:

```
10 X=30  
20 Y=10  
30  
40  
70 LOCATE X,Y
```



5.40 LPRINT/LPRINT USING

Estas dos clases de sentencias sacan los datos por la impresora. Funcionan de una manera muy similar a las sentencias PRINT (ver 5.51) y PRINT USING (ver 5.52).

```
LPRINT[expresión]<|;>expresión>[, ; ]]  
LPRINT USING<cadena de caracteres>;<expresión>  
{ <expresión> } [ ; ]
```

Para las operaciones de las sentencias, vea las sentencias PRINT y PRINT USING respectivamente.

5.41 LSET/RSET

Cuando se utiliza un fichero directo; la sentencia LSET/RSET envía los datos dentro del buffer antes de que los datos se escriban en el fichero.

```
LSET<variable de cadena>=<expresión de cadena>  
RSET<variable de cadena>=<expresión de cadena>
```

Izquierdo (Letra)
Derecho (Número)

- o <variable de cadena> especifica el nombre del área del buffer y es definida por la sentencia FIELD (ver 5.18).
- o <expresión de cadena> es el dato que se ha de escribir en el fichero y reemplaza la <variable de cadena> del área del buffer comenzando desde el final izquierdo por la sentencia LSET o desde el final derecho con la sentencia RSET. Cuando el número de caracteres de la <expresión de cadena> excede el valor definido por la sentencia FIELD, se desprecia el resto. Cuando es menor que el valor definido, los espacios se llenan.

- Cuando se utilizan datos numéricos, la función de conversión carácter-numérico (MKI\$, MKS\$, o MKD\$) es especificada antes que la anterior a la sentencia LSET/RSET para así convertir los datos numéricos a datos de cadena.
- Para el manejo de los ficheros del disco, vea el Capítulo 8.

Ejemplo:

```

OPEN "R",#1, "CUSTFILE",30
FIELD#1, 10 AS A$, 20 AS B$

X$=CUSTOMER-1"
Y$="12345"

LSET A$=X$
RSET B$=Y$

PUT#1, N

```

5.42 MERGE

Esta sentencia funciona intercalando el programa en memoria con el programa en el disco.

MERGE<nombre del fichero>

- El <nombre del fichero> indica que el fichero del programa está salvado en el disco, el cual deberá ser intercalado con el programa en memoria.
- Cuando el programa con el <nombre del fichero> especificado tiene los mismos números de líneas que el actual programa, estos números de línea machacarán a los existentes.
- El control vuelve al Modo BASIC cuando la sentencia MERGE ha sido ejecutada.

Ejemplo:

MERGE "PRGM-A"

5.43 MID\$

La sentencia MID\$ modifica una parte de datos de cadena con otra cadena de caracteres.

```
MID$(<variable de cadena>,<expresión numérica 1>
      [<expresión numérica 2>])=<expresión de cadena>
```

- o <La primera expresión numérica> y los demás caracteres de la <variable de cadena> son reemplazados con la primera por los caracteres de la <segunda expresión numérica> de la expresión de cadena.
- o Cuando se omite la <expresión numérica 2>, todos los caracteres de la <expresión de cadena> deberían utilizarse para reemplazamiento.
- o La longitud (número total de caracteres) de la <variable de cadena> no cambia.

Ejemplo:

```
A$="ABCDEFGHIJ"
B$="12345"
```

```
MID$ (A$, 5, 3)=B$      (A$="ABCD123HIJ")
```

```
MID$ (A$, 3)=B$      (A$="AB12345HIJ")
```

```
MID$ (A$, 8)=B$      (A$="ABCDEFG123")
```

5.44 NAME

Un nombre de fichero puede cambiarse con NAME.

```
NAME <antiguo nombre de l fichero> AS <nuevo nombre del fichero>
```

- o <El antiguo nombre> del fichero es reemplazado por el <nuevo nombre del fichero>
- o <El antiguo nombre> del fichero debe existir en el <disco> pero no el nuevo nombre.

Ejemplo:

```
NAME "FILE-A" AS "DATA-1"
```

5.45 NEW

NEW borra el programa en memoria.

NEW

La sentencia NEW siempre debe ejecutarse cuando se entra un nuevo programa desde el teclado.

El control vuelve al Modo BASIC después de la ejecución de la sentencia NEW.

5.46 ON ERROR GOTO

La sentencia ON ERROR GOTO

ON ERROR GOTO<número de línea>

- o <número de línea> indica el destino después de suceder un error. Las líneas que suceden a la de destino se pueden considerar como la rutina de manejo de errores.
- o La sentencia ON ERROR GOTO puede escribirse en cualquier sitio del programa tantas veces como desee el usuario. Cuando aparece un error, entonces el control salta al número de línea del último ON ERROR GOTO ejecutado.

Cuando aparece un error en el programa que no tiene la sentencia ON ERROR GOTO, entonces el sistema visualiza el mensaje de error y envía el control al Modo BASIC. Por el contrario, escribiendo la sentencia ON ERROR GOTO, el usuario puede determinar el control para el proceso erróneo del programa.

Esta instrucción no funciona contra el error durante la ejecución de la rutina de error. En ese caso, el sistema visualiza el mensaje de error y para el programa.

Cuando se entra la sentencia ON ERROR GOTO 0, la rutina que contiene el error resulta no válida. Es decir, se cancela la condición de salto según el error.

Ejemplo:

```
ON ERROR GOTO 800  
•  
• A=B/C  
•  
•  
END  
800 IF ERR=1 THEN C=1:RESUME  
IF ERR= ...  
•  
•
```

5.47 ON GOSUB/ON GOTO

Esta es una instrucción de ramificación condicional que sirve para especificar la designación con el valor en particular en el programa. Cuando se ejecuta, el control vuelve a la sentencia que sigue a la sentencia ON GOSUB a través de la instrucción RETURN de la subrutina parecida a la GOSUB.

```
ON<expresión numérica>GOSUB<número de lÍnea>{<número de lÍnea>}  
ON<expresión numérica>GOTO<número de lÍnea>{<número de lÍnea>}
```

- El valor computado de la <expresión numérica> especifica la lÍnea a la cual el control salta. El número de lÍnea debe escribirse con una constante de enteros.
- El valor de la <expresión numérica> debe estar entre el rango de 1 y (el número del <número de lÍnea>). Cuando incluye la parte fraccionaria, se redondea automáticamente (contando las fracciones sobre 5 y despreciando el resto). Cuando es negativo o se excede de 255, entonces aparece un error.
- Cuando el valor de la <expresión numérica> es 0 o se excede del <número de lÍnea>, esta sentencia no se ejecutará y el control va hacia la siguiente sentencia.

- o En el caso de la sentencia ON GOSUB el <número de linea> indicaría la linea de comienzo de la respectiva subrutina.

Ejemplo:

A=3
B=2

ON A+B GOTO 110, 120, 130, 140, 150, 160

(A+B=5, es decir el control pasa a la 150 el cual es el quinto número.)

5.48 OPEN

Para usar los ficheros del diskette, OPEN abre los ficheros de datos en el disco. Para más información vea el Capítulo 8.

OPEN<modo>, [<número del fichero>,<nombre del fichero> [<longitud del registro>]]

Uno de los siguientes cuatro caracteres es asignado a <modo>:

- (1) "O" (Salida) Modo de salida de ficheros secuenciales
- (2) "I" (Entrada) Modo de entrada de ficheros secuenciales
- (3) "A" (Apéndice) Modo de añadir a de los ficheros secuenciales
- (4) "R" (Aleatorio) Modo entrada/salida de ficheros directos

- o <El número de fichero> debe estar entre el rango de 1 y el número especificado del fichero en el momento de inicialización del BASIC. (Vea 1.1) La expresión numérica puede ser escrita por el <número del fichero>
- o <El nombre del fichero> especifica el fichero que hay que utilizar. (Ver 8.2)
- o <La longitud del registro> es el número de bytes para un registro de datos. Su especificación sólo se necesita en el modo "R". La longitud del registro se especifica libremente entre el rango de 1 y 128 bytes. Cuando se omite, se asume que el valor por defecto es "128".

- o Los ficheros secuenciales deberían ser abiertos después de que la operación de escritura y lectura haya sido seleccionada. Para los ficheros directos, ambos modos son válidos escribiendo una sola vez la sentencia OPEN.
- o El modo "A" indica el modo añadido de datos al fichero secuencial. Cuando está especificado, se pueden escribir de forma continua nuevos datos a continuación de los existentes en el fichero.
- o Cuando se abre el fichero, el nombre del fichero concuerda con su número y se le puede recordar implemente escribiendo el número del fichero en otras sentencias. Esto se mantiene hasta que el fichero se cierra. Cuando está cerrado, el usuario puede especificar el número del fichero con otro nombre y así abrir el nuevo fichero.
- o El fichero que se utiliza para entrar en el modo "I" o "R" puede ser abierto en duplicado con diferentes números de línea. Un fichero puede abrirse en modo "I" o "R" para entrar en más de un número de fichero a la vez. Sin embargo, para salir, un fichero debe abrirse en un sólo número de fichero. El fichero utilizado en modo de salida sólo puede ser abierto con el número de línea particular.

Ejemplo:

- o OPEN"I", 1,"DATA-1"
- o OPEN"O", 3,"FILE-A"
- o OPEN"R", N,"TRANS.001",80

5.49 OPTION BASE

Determina el valor mínimo del subíndice que se aplica a una matriz.

OPTION BASE n

- o "n" puede ser 0 ó 1.
- o Cuando la declaración de la matriz está realizada por la sentencia DIM, los elementos que van del 0 hasta el número declarado se reservan normalmente. Cuando se declara DIM A(20), hasta 21 elementos (A(0) a A(20)) se reservan en memoria. OPTION BASE se usa en el caso de que el usuario quiera reducir el número de elementos (20 elementos entre A(1) y A(20)).
- o Sólo se escribe una sentencia OPTION BASE en un programa. Se necesita ejecutarla antes de cualquier declaración DIM. (Las matrices con Base 0 y 1 no deben usarse mezcladas en un programa).
- o Igualmente, cuando los programas son concatenados por la sentencia CHAIN, las matrices de dichos programas deben tener la misma base de 0 ó 1.

5.50 OUT

Envía los datos de un byte al port de salida del procesador.

OUT<expresión numérica 1>,<expresión numérica 2 >

- o <La expresión numérica 1>produce el número del port con un entero que está entre 0 y 65535.
- o <La expresión numérica 2>indica los datos que hay que enviar, cuyo valor será un entero entre 0 y 255.

5.51 POKE

POKE escribe los datos de 1 byte dentro de la dirección de memoria especificada.

POKE<expresión numérica 1>,<expresión numérica 2 >

- o <La expresión numérica 1>especifica la dirección de la memoria en la cual están escritos los datos. El valor computado debe estar entre el rango de 0 y 65535.
La expresión numérica 2 especifica los datos a escribir y deben estar entre el valor de 0 y 255.
- o <Cuando ambas expresiones resultan en números reales, se redondea automáticamente (contando en fracciones sobre 5 como unidad y desentendiéndose del resto). Si se excede del rango, aparece un error.

Ejemplo:

POKE&HA000,&HFF
(FF se entra en la Dirección A000.)

FOR I=0 TO &HFF
POKE &H8000+I,&H20
NEXT

(Se entra un código de espacio desde 8000 a 80FF.)

5.52 PRINT

- o PRINT es una instrucción para dar acceso a la pantalla.

```
PRINT[expresión];[expresión];]
```

- o El valor computado de <expresión> sale por pantalla usando los numéricos o caracteres.
- o Cuando existe en la sentencia, un número plural de <expresiones> cada uno de ellos debe ser delimitado por una coma o punto y coma.
- o Cuando es una coma:
una línea de la pantalla (80 caracteres) se divide en 5 zonas de 16 caracteres cada una de ellas, y el valor valorado de <expresión> después de la coma aparece en la siguiente zona.
- o Cuando el limitador es un punto y coma:
el valor computado de la expresión después de un punto y coma aparece a continuación del valor de la expresión anterior.
- o Cuando se omite la coma o punto y coma al final de la línea,
 - o Se envía <CR> al final de la sentencia PRINT en la pantalla;
el Cursor vuelve y se fija a la izquierda del final de la línea; y
también queda determinada la línea en la pantalla.
- o Cuando existe la coma o punto y coma al final de la línea,
<CR> no se envía a la pantalla; y
en el caso de que se use una coma, la salida en pantalla continua desde la siguiente zona, o, si se usa un punto y coma, la pantalla continua inmediatamente después de la parte anterior.
- o Cuando se omite en la sentencia la parte que viene a continuación de <expresión>, sólo se ejecuta el avance de línea (aparece una línea en blanco).

Ejemplo:

```
>PRINT 10, 100, 0.1, -20
    10   100   0.1   -20
>PRINT 10;100;0.1;-20
    10 100 0.1 -20
>A=10: B=20: PRINT "A=";A, "B=";B
A= 10   B= 20
```

5.53 PRINT USING

Sirve para dar visualización en pantalla con el formato editado.

PRINT USING<expresión de cadena>;<expresión>, ;<expresión>[, :]

- El valor valorado de <expresión> sale por la pantalla en el formato especificado con <expresión de cadena>. El efecto de una coma (,) o punto y coma (;) es igual al de la sentencia PRINT.

Los formatos que <expresión de cadena> especifica son de la siguiente manera. <LF> no puede incluirse en el valor de <expresión de cadena>

- (1) cuando <expresión> es la expresión de cadena (salida del campo de caracteres):

- 1) "!" Visualiza el primer carácter del valor de expresión.

Ejemplo:
10 A\$="TOKYO"
20 B\$="JAPAN"
30 PRINT USING"!";A\$;B\$;"CCC"
RUN
TJC

- 2) "&#&" = 5 carácteres Visualiza hasta el carácter (n+2) del valor. Cuando los dígitos de los caracteres de expresión son menores que (n+2), se asignan espacios en blanco al resto.

Ejemplo:
10 A\$="TOKYO"
20 B\$="JAPAN"
30 PRINT USING "&#&";"CCC";A\$;B\$
RUN
CCC TOKYJAPA

- 3) Cuando <expresión> es la expresión numérica (salida del campo numérico):

- 1. #

Especifica la posición del puntero para la salida de caracteres.

Los caracteres válidos se visualizan empezando desde el final derecho de la línea en pantalla, y cuantos más espacios existan en la parte izquierda, los espacios en blanco deben ser tecleados.

Los caracteres numéricos pueden escribirse en el campo, y si se exceden, entonces aparece un error.

"##.##"

2. .(punto)

Un punto de fracción puede incluirse en los numéricos.

Ejemplo: >PRINT USING"##.##";0.536,3.1416,23.1

0.54 3.14 23.10

3. +,-

Los signos ("+" y "-") pueden escribirse en el prefijo o sufijo de los numéricos.

Ejemplo:

>PRINT USING"+ . ";230,-5,123.45
+230.0 -5.0 +123.5

>PRINT USING" . +";230,-5,123.45
230.0 +45.0- 123.5

>PRINT USING" . -";230,-5,123.45
230.0 45.0- 123.5

Cuando se realiza la especificación de un signo en el prefijo y sufijo, el primero toma la precedencia del sufijo y un carácter ocupa el lugar del signo en el sufijo.

PRINT USING"+# . +";-12.5
-12.5+

"#" se asigna al carácter en blanco para indicar el número de dígitos que se han de reservar en el campo.

Ejemplo:

>0 FOR I=0 TO 150 STEP 50
>0 PRINT USING"**#";:NEXT
RUN
***0
**50
*100
*150

"\\" puede escribirse en el prefijo del numérico. E incluyendo "\" los dígitos especificados quedan reservados en el campo.

Ejemplo:

>PRINT USING"\\";50,100,-500
40\50 40\100 -\500

5. //

isolo dos **

6. **

Tanto "##" como "###" pueden ser escritos. El numérico en un cierto número de dígitos, en el que se incluyen los tres dígitos para **, está salvado en el campo.

Ejemplo: >PRINT USING"##";50,1000,-12.6
 ***\50 *\1000 **-\13

7. ,(coma)

La coma puede ser escrita. Si precede al punto de fracción (punto), aparece en cada cuatro dígitos. Si está escrita después, se considera como un carácter. El cierto número de dígitos son reservados en el campo incluyendo las comas.

Ejemplo: PRINT USING"###,.;#";1234567.8
 1,234,567.80
 PRINT USING".,#";1
 1.00,

 PRINT USING"##,";1234567
 1,234,567

8.

Especifica el formato para la representación del exponente. El numérico en el número validado de dígitos aparece comenzando desde el final izquierdo, y la parte del exponente queda escrita de acuerdo a esto. Como el numérico no incluye "+" en el prefijo, o, "+" o "-" en el sufijo, entonces en la parte izquierda del punto de fracción debería escribirse "-" o un espacio.

Ejemplo: >PRINT USING".###";12345,-12345
 /.1234E+05 -.123E+05
 >PRINT USING".##00-";-0.00567
 .5670E-02-

Si los métodos 8 y 7 se toman concurrentemente, entonces la coma no es válida. O aparece un error cuando se toma el método 8 junto con " ".

Cuando otros caracteres que no sean los de arriba mencionados se escriben en la cadena de la especificación del formato, aparecen tal cual están escritos.

Ejemplo: >PRINT USING"- 3.56

5.54 PRINT / PRINT USING

Estas dos clases de sentencias tienen el mismo tipo de operación que la sentencia PRINT/PRINT USING pero en vez de imprimir en la pantalla, se graba en el correspondiente fichero de disco.

```
PRINT#<número de fichero>,[USING<cadena de Caracter>,<expresión>,<expresión>]
```

- o <El número> de fichero especifica el número bajo el cual el fichero se abrió para ser visualizado.
- o El fichero con el número de fichero especificado debe ser abierto en Modo "O" (Modo de Salida de Fichero Secuencial).

Ejemplo:

```
OPEN#1,"FILE.DAT"
```

```
PRINT#1,A,B,C
```

.

5.55 PRINT% / PRINT% USING

Estas dos clases de sentencias tienen el mismo tipo de operación que la sentencia PRINT/PRINT USING pero el periférico en el cual aparecen los datos es el port de comunicaciones serie del RS-232C.

```
PRINT%<número de port>,[USING<cadena de caracter>,<expresión>,<expresión>]
```

- o <número del port> será 1 ó 2 el cual coincide con el conectar LINE-1 O LINE-2.

Ejemplo:

```
PRINT%1,USING"##.",X";A,B,C
```

5.56 PUT

La sentencia PUT instruye para grabar dentro de un fichero directo. Con esta especificación, los datos que han sido salvados en el buffer deben ser entrados en el fichero directo.

```
PUT#<número de fichero>,<número de registro>]
```

- o <El número de fichero> especifica el número bajo el cual el fichero fue abierto para la salida en pantalla. El fichero con el número de fichero especificado debería ser abierto en el Modo Directo.

- o <El número de registro> indica cual es el registro que salva los datos que se han de entrar. Cuando se omite esta especificación, se añade un 1 al número de registro de la sentencia PUT o GET que ha sido ejecutada anteriormente. El número de registro especificado debe estar entre el rango de 1 y 32767.

Ejemplo: OPEN"R", 1,"CUSTFILE.DAT",30
FIELD 1, 10 A\$ AS, 20 AS B\$

X\$="CUSTOMER-1"
Y\$="10001"

LSET A\$=X\$
RSET B\$=Y\$

PUT 1, REC%

5.57 RANDOMIZE

La sentencia RANDOMIZE modifica la secuencia de números seudo-aleatorios que está creada por la función RND, y produce otra secuencia.

RANDOMIZE<expresión>

- o La valoración de <expresión> debe dar como resultado un entero desde -32767 a 32767.
- o La diferente matriz del número aleatorio está creada después de la ejecución de la sentencia RANDOMIZE a través de la valoración de la expresión.

5.58 READ/DATA/RESTORE

Las sentencias READ y DATA se combinan para su ejecución; la constante especificada en la sentencia DATA es asignada a la variable especificada en la sentencia READ. La sentencia RESTORE re-especifica el puntero para mostrar los datos de la sentencia DATA.

READ<variable>{<variable>
DATA<constante>{<constante>}

- <variable> es la cadena o datos numéricos y corresponde a cada una de las constantes de la sentencia DATA. Un número plural de las sentencias READ pueden escribirse para una sentencia DATA o viceversa. Las variables de la sentencia READ se leen ordenadamente desde el principio hasta el final, y en la que cada una de las variables corresponde a cada una de las constantes. Cuando el número de variables de la sentencia READ excede al de la constante de la sentencia DATA, entonces aparece un error. Cuando el número de las constantes excede al de las variables, se desprecia el resto de datos de la sentencia DATA.
- Las sentencias DATA pueden escribirse en cualquier lugar del programa tantas veces como deseé el usuario. La sentencia READ lee las constantes de las sentencias DATA en números de línea secuencial.
Cuando la variable en la sentencia READ son datos numéricos y la constante correspondiente en la sentencia DATA son datos de cadena, entonces aparece un error.
- Cuando la variable y la constante son datos numéricos pero sólo sus tipos son diferentes, la constante se convierte en variables del mismo tipo y los lee.
- Cuando la conversión resulta imposible, aparece un error.
Si el usuario quiere escribir comas (,), puntos y coma (;) o espacios entre la constante de caracteres, la constante debería encerrarse entre comillas (" ") .
- El usuario no necesita utilizar las " " para la constante de caracteres en la cual no existe ningún delimitador.
Las constantes en la sentencia DATA se usan secuencialmente conforme van apareciendo, sin embargo, la sentencia RESTORE puede cambiar la secuencia para la operación de lectura.

— RESTORE[<número de línea>]

Cuando se ejecuta la sentencia RESTORE, la sentencia DATA, después de la cual le corresponde ser ejecutada la sentencia READ, tiene el número de línea especificado en la sentencia RESTORE. Cuando se omite el número de línea, se asume que el número menor de línea corresponde a la sentencia DATA. Si este número de línea especificado no se encuentra entre las sentencias DATA, el puntero de lectura se determina a la primera aparición de la sentencia DATA más adelante que esa línea.

Ejemplo:

(1) DIM A(10)

FOR I=1 TO 10:READ A(I) :NEXT

DATA 1,2,5,10,3,7,15,8,3,4

A(1)=1
A(2)=2
A(3)=5
.
.
A(10)=4

(2)

10 READ A,B,C,D,E,F

20 RESTORE 60

30 READ G,H,I

50 DATA 10,10

60 DATA 20,20

70 DATA 30,30

A=10
B=10
C=20
D=20
E=30
F=30
G=20
H=20
I=30

5.59 REM

La sentencia REM indica que el comentario ha de ser entrado en el programa fuente en BASIC. No existe ninguna observación en relación a la ejecución del programa.

REM<comentario>
o
'<comentario>

- Se puede escribir en el comentario cualquiera de los caracteres útiles.

- Se puede escribir cualquier carácter en el <comentario>, si es que se admite en la lista de la Sección 2.1. El sistema ejecuta continuamente el programa despreciando el comentario.
- Cuando se lee la declaración REM en la Sentencia Múltiple, toda la parte empezando con REM y finalizando con <CR> es tomado como un comentario.
- Se puede usar un apóstrofe en lugar de REM.
- El control se va a la sentencia REM, sin embargo, el sistema realmente ejecuta la sentencia otra que REM que primero puede aparecer en las líneas posteriores.

Ejemplo:

```

10 REM***PRGM-1***  

20 ' 1981.1.1 ..  

30 '  

40 !;por G. Lyall  

50 '  

60 DIM A$(100)  

.  

.

```

5.60 RENUM

Todos los números de línea que están en memoria pueden ser modificados usando el comando RENUM.

RENUM[(<nuevo número de línea>), (<antiguo número de línea> ,<incremento>)]

- El <nuevo número de línea> es el primer número que hay que renovar. El <antiguo número de línea> indica la línea en la cual la operación de reenumeración debe empezar.
- <incremento> muestra el intervalo para la reenumeración.
- Cuando se omiten el <nuevo número de línea> y el <incremento> se asume que el valor por defecto es de 10 para cada uno de ellos. Cuando se omite el <antiguo número de línea>, la reenumeración comienza desde la primera línea que está almacenada en la memoria.
- Sigue un error cuando:
 - (1) no existe ningún programa en la memoria para ser reenumerado;
 - (2) los nuevos números de línea exceden de 65535;
 - (3) la secuencia del flujo del programa se cambia por la sentencia RENUM; y
 - (4) Se le asigna 0 al <incremento>.

- El número de caracteres de una línea puede pasar de 255 usando la sentencia RENUM. El programa puede funcionar y ser listado. Sin embargo, cuando la línea se modifica en el Modo de Edición en Pantalla, y si la longitud de la línea es de 255 caracteres, aparece un error.
- Entonces se necesita del usuario para modificar la línea que ha de ser de 255 caracteres.

Ejemplo:

<u>Antiguo número de línea</u>	<u>RENUM</u>	<u>RENUM 100</u>	<u>RENUM 70,52,20</u>
10	10	100	10
30	20	110	30
50	30	120	50
52	40	130	70
56	50	140	90
70	60	150	110
90	70	160	130

5.61 RESET

La sentencia RESET instruye el proceso a realizar antes de sacar el disco de la unidad de discos.

RESET

Cuando se ejecuta la sentencia RESET, el sistema cierra todos los ficheros abiertos, escribe los datos del índice en la pista del índice y reinicializa el sistema.

La sentencia RESET debería especificarse cuando el usuario saca el disco mientras que el programa no ha terminado todavía.

Sino, los datos correctos no podrán ser leídos otra vez fuera del disco.

5.62 RESUME

El control vuelve desde la subrutina que maneja el error al programa principal usando la sentencia RESUME.

RESUME [NEXT<número de línea>]

- La sentencia RESUME se usa junto con la sentencia ON ERROR GOTO. (Cuando se omite una de ellas, aparece un error). Si aparece un error, el control salta a la subrutina que maneja el error a través de la sentencia ON ERROR GOTO (ver 5.46). Cuando ésta subrutina ha terminado, vuelve a la línea del programa principal, cuyo número está especificado en la sentencia RESUME.
 - Cuando el número de línea no está especificado, el control vuelve a la sentencia por la cual ha sucedido el error. Cuando NEXT está escrito, vuelve a la siguiente sentencia errónea, y cuando el <número de línea> está especificado, va a la línea que contiene el número especificado.

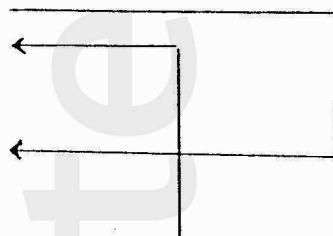
Ejemplo:

100 ON ERROR GOTO 1000

200 (Sucede un error)
210

1000 IF ERR=

1200 RESUME NEXT



5.63 RUN

La sentencia RUN lleva a cabo la ejecución de un programa.

- (1) RUN [*número de línea*] _o
- (2) RUN *número de fichero* ,R

- El formato (1) es para llevar a cabo la ejecución de un programa que ha sido grabado en memoria. Cuando <número de línea> es especificado, la ejecución comienza con la línea especificada, y cuando se omite, empieza con la primera línea del programa.
 - El formato (2) es para grabar el programa con el nombre de fichero especificado desde el disco a la memoria para su ejecución. Cuando la opción "R" es especificada, el sistema graba y ejecuta el programa de datos guardando los ficheros abiertos.

RUN
RUN 330
RUN "NOM.A"
RUN "NOM.A", R

Ejemplo:

- RUN
- RUN 300
- RUN "PRGM"
- RUN "PRGM-A",R

5.64 SAVE

La sentencia SAVE instruye la operación de almacenamiento del programa desde la memoria al fichero del disco determinado.

SAVE <nombre de fichero> '[, A|P|Q]

- <nombre de fichero> es asignado al programa. (Ver 8.2)
Puede llamarse con la sentencia LOAD (ver 5.38) y grabarse otra vez en memoria.
- Como A indica el modo en código ASCII, el programa debería escribirse en el disco tal como es. Cuando P es especificado, el programa se convierte al código intermedio para escribirlo dentro del fichero del disco. Si no se realiza ninguna especificación en la sentencia, P es asumida.
- Cuando un programa salvado con Q es grabado en la memoria, es imposible de realizar un listado de un programa y salvarlo (guarde el secreto del programa y la copia de seguridad).
- A, P, Q son opcionales.

Ejemplo:

- SAVE "PRGM-1"
- SAVE "PRGM-1",A
- SAVE "PRGM-A",Q

5.65 SET

La sentencia SET define el atributo para los discos del fichero.

SET#<número de fichero>|<nombre de fichero>|<unidad>, <símbolo del atributo>

- Los atributos del fichero son divididos en las dos siguientes formas:

W Protección de escritura (Sólo Lectura)
 Prohibe la operación de escritura.

S Sistema (Invisible)
 No todo se puede leer fuera del fichero por la sentencia FILE/LFILES, para la cual se determina el atributo "S".

- El <número del fichero> especifica el fichero abierto, y el <nombre del fichero> especifica el nombre de ese fichero en particular. <unidad> muestra el nombre de una unidad de discos, es decir, A: 6 B: (C: 6 D: también pueden utilizarse cuando se instalan unidades optionales). Especificando estos tres elementos, el atributo puede ser determinado.
- El atributo puede ser reinicializado por la sentencia SET con la especificación "N".

Ejemplo:

```

SET #1,W
SET "FILE.DAT",W
■ . SET "FILE.DAT",N (Reinicialización del atributo)
. SETA:,W
. SET#3,S
. SET B:,N      (Reinicialización del atributo)

```

- Los contenidos del fichero pueden ser dañados si el comando SET se ejecuta cuando un fichero está abierto.

Asegúrese de ejecutar el comando SET después de cerrar el fichero.

5.66 STOP

Mediante el comando STOP, el sistema detiene la ejecución del programa y el control retrocede al control del Modo BASIC.

STOP

- El comando STOP puede escribirse en cualquier punto del programa y cuando se ejecuta, el sistema hace aparecer el número de línea en donde se ha detenido el procesamiento:
Breaking in line XXXX
Entonces, el control vuelve al Modo BASIC.
- Un programa detenido mediante la sentencia STOP puede reanudarse, desde la siguiente línea, usando la sentencia CONT (ver 5.8).
- El comando STOP no cierra los ficheros.

5.67 SWAP

Esta sentencia cambia el contenido de dos variables diferentes.

SWAP <variable 1> , <variable 2>

- El contenido de <variable 1> y el de la <variable 2> son intercambiables.

- El tipo de las dos variables debe ser el mismo.

Ejemplo:

- SWAP A,B
- IF A(N)>A(N+1) THEN SWAP A(N), A(N+1)

5.68 SYSTEM

La sentencia SYSTEM hace volver el control al Modo del Sistema Operativo.

SYSTEM

- Cuando se ejecuta la instrucción SYSTEM, el control vuelve al Modo del Sistema Operativo y los ficheros que estén abiertos, quedarán cerrados.

5.69 TINPUT

Esta es la instrucción de entrada de datos, a través del teclado. La sentencia INPUT hace aparecer los datos de entrada, haciendo mover el Cursor una columna hacia la derecha; por otro lado la sentencia TINPUT traslada los datos de entrada hacia la izquierda, manteniendo el Cursor tal como está. (La visualización de la línea empieza al final del lado derecho.) La sentencia TINPUT se usa, especialmente, en lo concerniente a la entrada de datos numéricos.

TINPUT [*<expresión numérica>*] [;] <variable>

*; Entrada y separado
(Nada) Entrada y aparte*

- Cuando va apareciendo los datos de entrada por la pantalla, la sentencia TINPUT especifica la posición del Cursor para que esta se convierta en el fin de la línea y empieza la visualización de la línea, desde el final del lado derecho. Los datos pasan una columna hacia la izquierda, cada vez que se pulsa un nuevo carácter. El Cursor no se mueve.
Cuando el primer carácter pulsado llega al final del lado izquierdo, sólamente se podrá entrar <CR> y todos los demás caracteres, incluso los ya escritos, no serán tomados en cuenta.
- Cuando la variable no concuerda con la constante, el sistema no se detiene ante este error, sino que espera otras pulsaciones borrando los datos entrados hasta entonces.
- Cuando se introduce un punto y coma (;) entre el TINPUT y <variable>, el <CR>, para mostrar el final de una línea, no aparece en la pantalla y el Cursor no se mueve. En el caso de que se omita el punto y coma, el Cursor retrocede hasta la siguiente línea.
- <expresión numérica> controla los dígitos de los datos que se van escribiendo, efectivo del 1-80.

- Para usar la sentencia TINPUT correctamente, el Cursor debe iniciarse con antelación en la posición que le corresponde, atribuyendo el número o caracteres de datos que han de ser entrados. (Con este propósito, use la sentencia LOCATE (ver 5.39) o la PRINT (ver 5.52))
- En los otros casos la sentencia TINPUT funciona de forma similar a la sentencia INPUT (ver 5.26).

Ejemplo:

- TINPUT A\$
- TINPUT; X

5.70 TRON/TROFF

Estas sentencias dan instrucciones a la inicialización y reinicialización del Modo Trace para la depuración del programa.

TRON [P]
TROFF

- Cuando se lleva a cabo la instrucción TRON (para entrar el Modo TRACE), todos los números de línea de un programa que habían sido ejecutados hasta aquel momento, aparecen por la pantalla, desde el principio de la especificación. Cuando se lleva a cabo la instrucción TROFF, el control vuelve al modo normal de ejecución.
- Incluso en el Modo Direct, puede desarrollarse la ejecución del programa a través de la instrucción TRON/TROFF.
- Cuando se escribe TRON P, el sistema pasa los números de línea a la impresora.
- Cuando se lleva a cabo la instrucción NEW aparece el Modo Trace

Ejemplo:

TRON
TRON P
TROFF

5.71 WAIT

El comando WAIT da instrucciones a la supervisión de las puertas de entrada de CPU (Unidad Central de Procesamiento).

WAIT <número de puerta> , <expresión 1> [, <expresión 2>]

- Cuando se procede con la instrucción WAIT, la ejecución del programa se detiene hasta que se crea un determinado modelo de bit de las puertas de entrada CPU, cuyo número queda especificado con <número de puerta>. Primeramente, se hace una suma exclusiva lógica de los datos de la puerta de entrada y del valor de <expresión 2>, luego se hace un producto lógico del resultado de la suma y el valor de <expresión 1>. En el caso de que el producto lógico se convierta en 0, se vuelven a leer los datos de la puerta de entrada. Estos procesos se van repitiendo. Cuando el producto lógico no es 0, el control del programa pasa al siguiente paso. En el caso de que se omita <expresión 2> se atribuirá el valor "0".
- Los resultados de evaluación de <expresión 1> y <expresión 2> deben ser los enteros entre 0 y 255.

5.72 WHILE/WEND

Estas sentencias dan instrucciones para que las sentencias se vayan desarrollando consecutivamente.

```
WHILE <expresión>
  ---
  WEND
```

- Cuando las condiciones dadas por <expresión> son efectivas, las sentencias hasta WEND son ejecutadas de forma repetida. La diferencia entre la sentencia WHILE/WEND y la de FOR/NEXT es que la valoración de las condiciones se hace antes de que las sentencias sean bucleadas.
- Si la evaluación de <expresión> no da como resultado 0 (la expresión es correcta), se ejecutan un grupo de sentencias desde la siguiente hasta la WEND. Cuando el control llega a la sentencia WEND, retrocede a la sentencia de WHILE para la re-evaluación de <expresión>. En el caso de que <expresión> sea afirmada, el bucle se vuelve a ejecutar. En el caso de que sea evaluado como 0 (la expresión no es afirmada), el control salta a la siguiente sentencia después de WEND.
- Las sentencias de WHILE y WEND pueden ser anidadas. La sentencia WEND puede ser anidada a la anterior sentencia WHILE, según la secuencia de línea.

Ejemplo:

```
WHILE A>0
  X=X+Y
  A=A-1
WEND
```

5.73 WIDTH

La sentencia WIDTH determina la longitud de la salida (el número de caracteres en una línea) de la impresora.

WIDTH [expresión]

- Cuando se ejecuta el WIDTH, el sistema alimenta la línea, cada vez que el número especificado de caracteres llene el espacio de la línea. El número de caracteres es aportado con el valor de evaluación de <expresión>.
- El valor de evaluación de <expresión> debe ser un entero desde 15 a 255. Cuando no es un entero, la parte de fracción debe ser redondeada (contando las fracciones 5 y por encima de 5 como una unidad, no teniendo en cuenta el resto). Si no está dentro de esta oscilación, aparecerá un error.
- Cuando no se especifica la longitud de una línea, mediante la sentencia WIDTH, para la impresión de una línea en SBASIC, se iniciará una línea de hasta 80 caracteres.
- La evaluación nos da como resultado 255, la longitud se considera ilimitada y el sistema no manda el código <CR> a la impresora. La posición de inicio de la impresora, en este caso, es de más de 255 y el 0 vuelve a la función LPOS.

Ejemplo:

- WIDTH 132
- WIDTH 65

*Máximo para C.I TO
136*

5.74 WRITE

Similar a la sentencia PRINT, el comando WRITE da instrucciones a la salida de datos en la pantalla. Sin embargo, cuando se especifica el comando WRITE, las comas quedan automáticamente colocadas entre los elementos de salida, y los datos de cadena aparecen entre comillas (" "), así mismo de forma automática.

WRITE [expresión {, expresión}]

- El valor de <expresión> aparece en la pantalla. Si existen varias <expresiones>, cada una de ellas queda delimitada por una coma pero sin seguirles ningún espacio. Cuando <expresión> es numérica, el cero queda suprimido.
- En el caso de <expresión> de cadena, la salida de la cadena de caracteres en la pantalla, queda encerrada, automáticamente, entre comillas (" ")

- Cuando <expresión> se omite, pasa a la siguiente línea. (En la pantalla, aparece una línea en blanco).

Ejemplo: >10 A=100

> 20 B=5

> 30 C\$="TOKYO"

> 40 D\$="JAPAN"

> 50 WRITE A,B,C\$+D\$

>RUN

100,5, "TOKYOJAPAN"

5.75 WRITE

El comando WRITE# instruye la salida de datos a los ficheros (secuenciales) del disco, y los datos se escriben en los ficheros del disco, en un formato similar al comando WRITE. Las comas, igual que las comillas, se añaden automáticamente, así pues, los delimitadores se escriben en los ficheros del disco muy fácilmente.

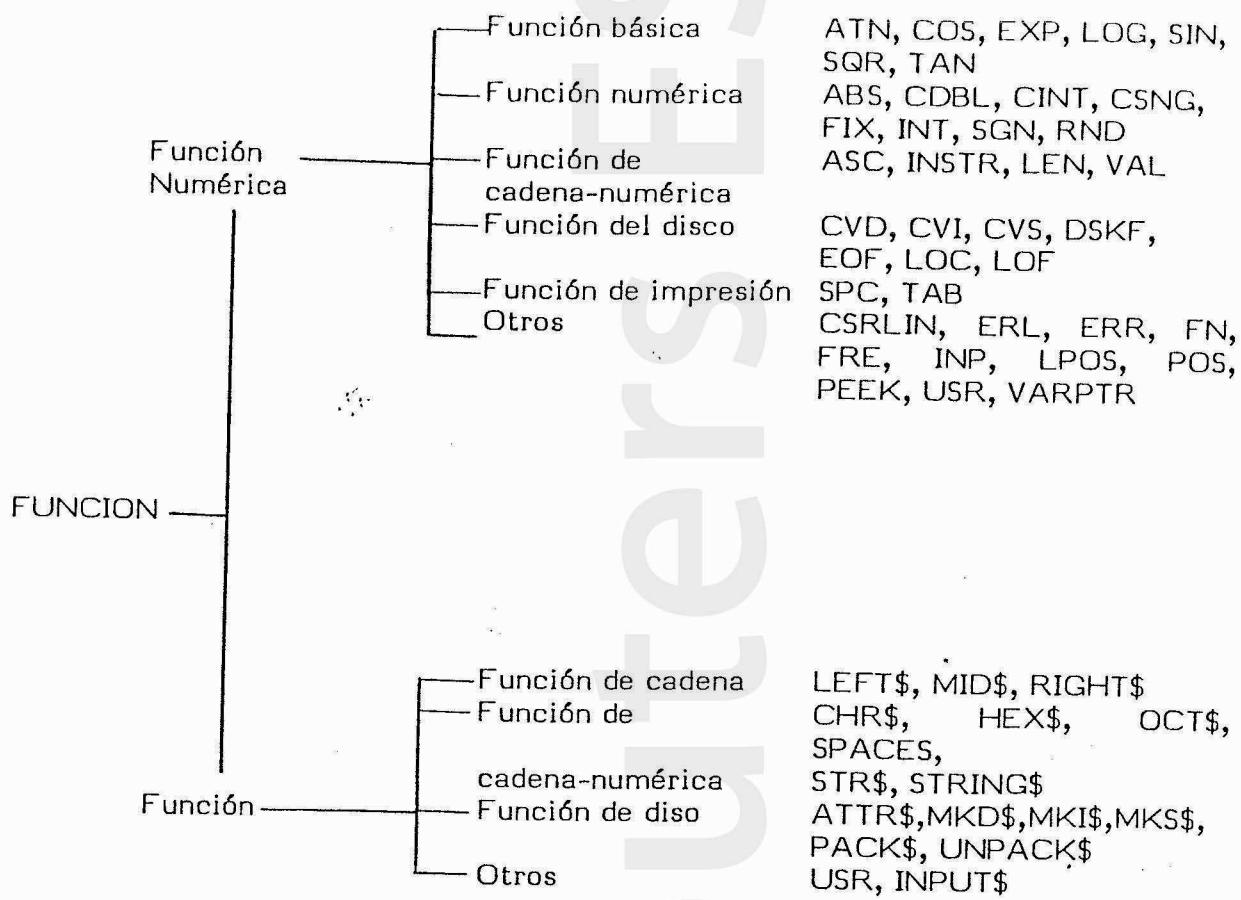
WRITE#<número de fichero>, <expresión>, {<expresión>}

El <número del fichero> indica que el fichero se abre para escribir.. El fichero que se ha de utilizar, debe haber sido abierto en Modo "O" ó "A".

Vea el capítulo 8, para los usos determinados de las instrucciones PRINT#, WRITE#, INPUT# y LINE INPUT#.

(6) FUNCTION

Hay dos clases de funciones, la función numérica (la función cuyo valor es numérico) y la función de cadena (la función cuyo valor se convierte en una cadena de caracteres). Algunas funciones tienen el parámetro(s), que se describen entre paréntesis "()" inmediatamente después del nombre de la función. Si la función tiene más de dos parámetros, cada uno de ellos quedará delimitado con una coma. Las funciones de cadena tienen "\$" en su nombre.



El valor de función y el parámetro de la función numérica son enteros, números reales de precisión simple o números reales de precisión doble. La operación en el sistema interno de la función básica se lleva a cabo en precisión doble y de esta forma, la precisión del valor de función de la función básica se mejora, siempre que el parámetro queda especificado en precisión doble. En la explicación de cada función, se emplean ciertos símbolos para la especificación de los parámetros. En los ejemplos de la columna derecha de la tabla, vemos la descripción práctica para la llamada.

Símbolos descriptivos	Significado
X, Y I, J	⟨expresión numérica⟩ ⟨expresión numérica⟩ de cada valor de evaluación se convierte en un entero. (En el caso de que se convierta en un número real, será redondeado de forma automática, contando las fracciones de cinco en adelante sin tomar en cuenta el resto.)
X\$, Y\$	⟨expresión de cadena⟩

Nº	Nombre de función y parámetro	Valor de Función	Ejemplo
1	ABS(X)	Valor absoluto de X	ABS(5*(-1))→5
2	ASC(X\$)	El valor del código de conversión JIS del primer carácter en X\$; cuando X\$ es la cadena nula, aparece un error.	ASC("ABC") → 65
3	ATN(X)	Arcotangente de X (función trigonométrica) (Radián)	ATN(3.14159 → 1.26263
4	ATTR\$(a) a:<nombre de unidad> #<número de fichero,> o<nombre de fichero>	Símbolo de atributo del fichero especificado (W, S, o cadena nula) *Ver 5.65	ATTR\$(B:) ATTR\$(#2) ATTR\$("FILE")
5	CDBL(X)	Transformación de X en número real de precisión doble.	CDBL(1.234)
6	CHR\$(I)	Caracter con un código "I" JIS	CHR\$(&H41) → "A"
7	CINT(X)	Transformación de X en entero (contando en fracciones sobre 5 y despreciando resto); cuando X<-32768 o X>32767, aparece un error.	CINT(2.5) → 3
8	COS(X)	Coseno de X (función trigonométrica) (El parámetro es radián.)	COS(3.14159)→1
9	CSNG(X)	Transformación de X en número real de precisión simple	CSNG (1.2345678) →1.23457
10	CSRLIN	Coordenada vertical(puntero de linea) de la actual posición del Cursor (1-24); la función POS para la coordenada horizontal (puntero de carácter)	LOCATE POS, CSRLIN+5 (El Cursor va a la 5ª linea)

11	CVD(caracteres de 8 bytes)	Número real de precisión doble del valor numérico de caracteres de 8 bytes (Transformación inversa de MKD\$)	
12	CVI(caracteres de 2 bytes)	entero del valor numérico de caracteres de 2 bytes (Transformación inversa de MKI\$).	Para leer los ficheros directos Para más información, ver del 6.36 al 6.38 y 8.10.
13	CVS(caracteres de 4 bytes)	Número real del valor numérico de caracteres de 4 bytes. (Transformación inversa de MKS\$)	
14	DSKF A (<nombre de unidad>)	Area en blanco del disco almacenado en la unidad especificada por <nombre de unidad> (kilobytes)	PRINT DSKF(B:) 128
15	EOF(<número del fichero>)	Cuando el fichero con el número especificado alcanza el EOF (fin del fichero), el valor se convierte en -1; en otros casos se asigna un 0.	IF EOF(3) THEN 200 (Cuando el fichero #3 alcanza EOF, el control va al 200. Sólo es válido para los ficheros secuenciales en modo "I").
16	ELR	Número de la línea donde aparece un error.	IF ERR=2 NAD ERL=100 THEN GOTO 1000.
17	ERR	Código de error.	
18	EXP(X)	ex	EXP(1)→2.71828
19	FIX(X)	Valor de X cuya parte de fracción no se toma en cuenta. (El punto de la fracción se omite sin tener en cuenta el signo).	FIX(3.25)→3 FIX(-3.8)→-3

20	FRE	Número de bytes que quedan en la memoria (número decimal)	PRINT FRE
21	HEX\$(X)	Valor hexadecimal de X	PRINT HEX\$(255) FF
22	INKEY\$	Caracter a escribir; si no se pulsa ninguna tecla, \$ se convierte en cadena nula.	10 IF INKEY\$="A" THEN 1000 ELSE 10 (El control se espera hasta que se pulse la tecla "A". Una vez se pulsa, va a la 1000)
23	INP(I)	Transfiere un byte del port E/S 1/0 del CPU. "I" indica el número del port.	
24	INPUT\$(I J%J)	Lee los caracteres (I) entrados desde el teclado, el disco o los ports en serie. #J es el número del fichero y %J es el número del port. Cuando sólo se escribe "I", se considera el teclado. Los datos de entrada no aparecen en la pantalla.	A\$=INPUT\$(5) X\$=INPUT\$(10, 2) Y\$=INPUT\$(1,%1)
25	INSTR([I],X\$,Y\$)	Localización de Y\$ en X\$; empieza desde el 1er carácter.	X\$="ABCDXXX DEFYY" Y\$="XXX" INSTR(X\$,Y\$) 4

26	INT(X)	Entero máximo que no excede X	INT(258.4)→ 258 INT(-301.8)→ -302
27	LEFT\$(X\$,I)	Caracteres a la izquierda de X\$ a I; si K0, y I>255, aparece un error	A\$="ABCDEF" LEFT\$(A\$,3)→ "ABC"
28	LEN(X\$)	Longitud de los caracteres	LEN("ABCDE")→ 5
29	LOC(<número de fichero>)	Fichero aleatorio:el siguiente nº de registro Fichero secuencial:número de registros secuenciasl escritos o leídos después de abrir el fichero.	LOC(1)
30	LOF(<número de fichero>)	Número de registros escritos o leídos en el fichero especificado. (1 registro=128 bytes)	LOF(1)
31	LOG(X)	Logaritmo natural de X	LOG(2)→ 0.693147
32	LPOS	Posición del cabezal para imprimir	
33	MID\$(X\$,I,J)	Caracteres J(número) sobre I en X\$; Caracteres entre I y un final, entonces J se omite.	A\$="1234567" MID\$(A\$,3,2)→ "34" MID\$(A\$,5)→ "567"

34	MKD\$ (<expresión>)	<expresión> se convierte en número de precisión doble. CVD para transformación inversa	
35	MKI\$ (<expresión>)	<expresión> se convierte en entero. CVI para transformación inversa.	Transforma el valor de la expresión en cadena de carácter. Usado principalmente para escribir en ficheros aleatorios.
36	MKS\$ (<expresión>)	<expresión> se convierte en número real. CVS para transformación inversa.	
37	OCT\$(I)	Cadena de caracteres de notación octal de I	OCT\$(256)→ "400"
38	PEEK(I)	Datos de memoria en dirección "I" (datos numéricos)	POKE &HA000, &HFF PRINT PEEK (&HA000)→255
39	PACK\$(X\$)	X\$ es una cadena de caracteres que indica el valor numérico de 0~9,+,-,.,D,E, y espacios. Crea una cadena de caracteres cuyo número de bytes es una mitad de los X\$. UNPACK\$ se utiliza para la transformación inversa.	LEN(PACK\$ ("123456"))→ 3 Usado para escribir en el fichero del disco.
40	POS	Coordenada horizontal (puntero del carácter) de la posición (1-80) del Cursor en memoria: la función CSRLIN es para la coordenada vertical (puntero de la línea).	

41	RIGHT\$(X\$,I)	Caracteres a la derecha de X\$ a I	A\$="ABCDEF" RIGHT\$(A\$,2)→ "EF"
42	RND (<expresión aritmética>)	Distribución uniforme de la cadena de números seudo-aleatorios de intervalo 0.1 Los diferentes números directos se crean dependiendo del valor de la expresión aritmética.	DIM A(100), B(100) FOR I=1 TO 100 A(I)=RND(1) B(I)=RND(2) NEXT (Los diferentes números directos son asignados a A y B).
43	SGN(X)	Signo de X: SGN(X)=-1 X<0 SGN(X)=0 X=0 SGN(X)=1 X>0	
44	SIN(X)	Seno de X (función trigonométrica) (Radián)	SIN(3.14159/2)→ 1
45	SPACE\$(I)	I (número de) espacios "como cadena"	
46	SPC(I)	I(número de) espacios de salida (sólo se usa en la sentencia PRINT/LPRINT) "como tabulación"	
47	SQR(X)	Raiz cuadrada de X	SQR(2)→ 1.41421

48	STR\$(X)	Transforma el valor numérico de "X" en cadena de caracteres. VAL(X\$) se usa para transformación inversa.	STR\$(100) → " 100"
49	STRING\$	I (número) caracteres "J" del código JIS o caracteres de arriba del X\$ se escriben repetidamente en una cadena de caracteres.	PRINT STRING\$(3,"ABC") → "AAA" STRING\$(4,&H43) → "CCCC"
50	TAB(I)	Espacio de tabulación para imprimir desde la posición del carácter "I" contando a partir del final izquierdo. Sólo es efectivo en la sentencia PRINT/LPRINT. En la sentencia PRINT, esta función también funciona en las posiciones de la izquierda que hay a continuación de la posición del Cursor.	PRINT TAB(20); "A" Sólo con cursor y no en blanco pero de espaciado.
51	TAN(X)	Tangente de X(función trigonométrica) (Radián)	TAN(3.14159/4) → 1
52	UNPACK\$(X\$)	Recupera la cadena de caracteres a su estado original.	Usado para leer desde los ficheros.
53	USR<n>(X)	Función del usuario definida en la sentencia DEF USR (ver 5.12)	
54	VAL(X\$)	Valor numérico que muestra la cadena de caracteres del X\$ (transformación de datos de los caracteres numéricos; STR\$ es para la transformación inversa).	VAL("12345") → 12345 VAL("123AAA 56") → 123

55	<p>VARPTR <variable></p> <p>VARPTR (#<número de fichero>)</p>	<p>Dirección donde se almacena la <variable> cuando los números sobrepasan los 8000H se convierten en negativos, es mejor transformar los números usando la función HEX\$. (Cuando la variable es un valor numérico, aparece la dirección del byte más pequeño. Cuando la variable es un dato de carácter, aparece la dirección del byte más pequeño del índice de la cadena de caracteres. Ver pág. 42).</p> <p>Dirección del byte más inferior del área del buffer en el fichero que tiene el número de fichero especificado.</p>	<p>HEX\$Δ (VARPTR(AS))</p> <p>HEX\$Δ (VARPTR(#2))</p>
56	FN<nombre>	Función del usuario definida por la instrucción DEF FN (5,12).	

7. EDICION DE PROGRAMAS

7.1 Edición de Programas

Este capítulo explica dos métodos de la edición de programas, que son:

- (1) Adición de una lÍnea
- (2) Corrección de una lÍnea

En el método (1), un programa en BASIC se graba en la memoria lÍnea por lÍnea tecleadas por el operador. En el método (2), el usuario invoca al programa, que ha sido cargado, en la pantalla y realiza modificaciones o cambia la secuencia del programa parcialmente.

7.2 Adición de Líneas

(1) Entrada de un programa

Cuando el computador visualiza en pantalla, en Modo BASIC, el signo del BASIC y el Cursor, (el cursor se verá en situación de destello a continuación del signo del BASIC) es decir que, cuando el sistema tiene inicializado el programa en BASIC o en Modo BASIC después de haber realizado las instrucciones necesarias, el operador puede ejecutar la entrada de una lÍnea en el programa.

Para la entrada de una lÍnea, él o ella deberá escribir primero el número de lÍnea, entonces la sentencia, y CR al final. Cuando se pulsa la tecla CR, la lÍnea queda grabada en la memoria, y el sistema revisa los Errores de Edición. (Ver Apéndice B).

El operador puede borrar un carácter pulsando la tecla DEL hasta que se pulsa CR. CE sirve para borrar una lÍnea completa (cuando el Modo AUTO funciona sólo el número de lÍnea es válido -igual como si se mantuviera pulsada la tecla CE. Si es por lo contrario, también se borra el número de lÍnea). Una lÍnea completa de programa se forma a partir del número de lÍnea hasta CR. La longitud de la lÍnea puede tener hasta 255 dígitos incluyendo un número de lÍnea y espacio(s).

El programa en la memoria se maneja con los números de lÍnea; puede ser listado o ejecutado con los números de lÍnea en secuencia. El número de lÍnea mayor siempre sigue al inferior, y así se va almacenando el programa de acuerdo a este orden, de manera que cuando se escribe, la lÍnea con su número correspondiente se coloca en su posición correcta. Si la lÍnea ya existe en memoria, cuyo número de lÍnea es el mismo que el entrado, el contenido se renueva. Cuando hay exceso de número de lÍneas del programa, el sistema envía el correspondiente mensaje de error.

(2) Borrar un programa

Las dos siguientes instrucciones llevan a cabo la operación de borrar del programa. Para más detalles, vea las secciones previamente mencionadas.

- (1) Borrado del programa entero Sentencia NEW (Ver Sección 5.45).
- (2) Borrado de una parte del programa Sentencia DELETE (Ver Sección 5.13).

Cuando el operador desea borrar sólo una línea en particular, debe escribirse su número de línea y CR.

(3) Listado del programa

Cuando el control está en Modo BASIC, el programa, que está almacenado en memoria, puede ser sacado por la pantalla o impreso total o parcialmente.

(1) Listado del programa por pantalla Sentencia LIST (Ver 5.37)

(2) Listado del programa por impresora Sentencia LLIST (Ver también 5.37).

Cuando el operador quiere parar temporalmente el listado, debe pulsar la barra de espacio del teclado. El listado se para inmediatamente. Y cuando se vuelve a pulsar otra tecla, el listado continua. Con esta operación, el usuario puede revisar el contenido del programa sin tener en cuenta la longitud. Si él o ella desea dejar de listar mientras el proceso de impresión está parado o en proceso, debe pulsar la tecla BREAK. Entonces el sistema detiene la impresión y el control vuelve al Modo BASIC.

(4) Salvar un programa

El programa que está en memoria puede ser salvado en disco usando la sentencia SAVE (ver 5.64). El programa que se ha pasado dentro del disco puede leerse en memoria usando la sentencia LOAD (ver 5.38). Para más detalles, vea el Capítulo 8.

(5) Mezcla de dos programas

Mediante la sentencia MERGE, el programa en memoria puede ser mezclado con otro programa que ha sido leído recientemente desde el fichero del disco. En Modo BASIC el operador escribe en la sentencia MERGE (5.42) el siguiente formato:

MERGE "PRGM-B" <CR>

Entonces el sistema lee en "PRGM-B" desde el disco y crea un nuevo programa mezclándolo con el programa ya existente en memoria. Cuando se incluye en "PRGM-B" el mismo número de línea, el contenido de la línea queda renovado tal como está en "PRGM-B".

(6) Operaciones AUTOmáticas y RENUMeración

Cuando se escribe, el sistema puede crear y visualizar los números de línea automáticamente. El comando AUTO (5.1) se utiliza para este cometido.

El operador asigna el comienzo del número de línea y el incremento a los parámetros del comando AUTO, así pues el sistema visualiza automáticamente los números de línea siguientes cada vez que una línea ha sido tecleada con un CR. (En el modo AUTO, el número de línea no se borra aunque la otra parte de la línea haya sido borrada por el uso de la tecla CE).

Para la total modificación de los números de línea, se ha de entrar la sentencia (5.60) RENUM (5.60). Con la sentencia, el operador puede fijar todos los números de línea existentes para hacer espacio entre los dos números diferentes para líneas adicionales.

7.3 Corrección de Línea

Las teclas **→**, **DEL**, **CR** y **BREAK** son necesarias para la edición de líneas para cambiar o corregirlas, duplicarlas, etc.

Ejemplo (1): Cambia el parámetro A al parámetro B.

Ready
>EDIT 100
100 PRINT "TOKYO=";A;"*"
100 PRINT "TOKYO=";B;"*"

Use **→** key. Entre tecla **B**.

Pulse tecla **DEL** seguida de **CR**.

Ejemplo(2): Duplica los contenidos de la línea 1000 a la línea 10.

Ready
>EDIT 1000
1000 PRINT "TOKYO"
10 **DEL** **DEL**

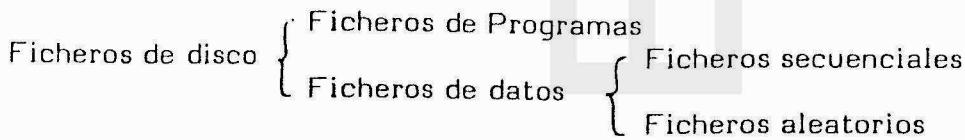
Cuando se pulsa **CR**, los mismos caracteres así como la línea 1000(**PRINT "TOKYO"**) quedan duplicados en la línea 10.

Nota: Si la línea 10 ya existe, el contenido de la línea 1000 queda duplicada en la antigua línea 10.

8. FICHERO DEL DISCO

8.1 Tipos de Ficheros

Se manejan los siguientes ficheros de disco.



Cada fichero tiene un nombre con que se maneja la operación para Invocar.

8.2 Nombres de Ficheros

Se asigna un nombre a un fichero determinado. La especificación del nombre del fichero se hace de acuerdo a las reglas que se mencionan a continuación:

- (1) Un nombre de fichero consiste de 11 caracteres (8 caracteres para la identificación del fichero y 3 para la extensión), y se coloca un punto (.) para delimitar las dos partes.

Δ,Δ,Δ,Δ,Δ,Δ,Δ,Δ,Δ,Δ,Δ

nombre del fichero

Δ,Δ,Δ

nombre del fichero extenso

Ejemplo:

TEST.BAS
TESTPRGM.BAS
DATAFILE.A1
FILE-ABC
SBASIC.COM

El nombre del fichero extenso no se define si no es necesario. Cuando se necesitan más de 8 caracteres para el nombre de un fichero, los caracteres que se encuentran entre el 9 y 11 son considerados como nombres de ficheros extensos. Los que sobrepasan la posición 12 son despreciados.

Ejemplo:

TESTPROGRAM-A → TESTPROG.RAM

El nombre del fichero extenso indica el grupo en particular al que pertenece el fichero. Como "COM" se usa como el nombre del fichero extenso de un programa de sistema y "BAS" como el de un programa en BASIC, el usuario es avisado para que no haga uso de estos nombres de ficheros extensos.

Cuando las palabras de la siguiente instrucción relacionada a los ficheros del programa se usan en Modo BASIC, pero cuando el usuario no escribe los nombres del fichero extenso, entonces el sistema añade automáticamente "BAS" a los nombres del fichero.

Inicialización del BASIC (ver 1.1)

CHAIN	(Ver 5.4) Load + Run de un 2º programa
LOAD	(Ver 5.38)
MERGE	(Ver 5.42) une programas.
RUN	(Ver 5.63)
SAVE	(Ver 5.64)

- (2) Un nombre de fichero debe empezar con un abecedario. Para el segundo y posteriores caracteres, pueden utilizarse cualquier alfanumérico y símbolos que hay a continuación.

#, \$, %, &, '(), +, -, /

Se pueden utilizar las letras situadas en la parte superior e inferior de las teclas, para ello el usuario las debe de escribir de manera que se distingan.

- (3) Cualquier tipo de caracteres (constante, variable o cadena) son válidos para escribir un nombre de fichero en un programa.

Ejemplo:

(1) SAVE "PRGM-A"	→ PRGM-A.BAS
(2) LOAD "PRGM-A"	→ PRGM-A.BAS
(3) MERGE "PROGRAM-A"	→ PROGRAM-.A--
(4) A\$="FILENAME" OPEN"O">#1,A\$	→ FILENAME.---?
↓	
(5) A\$="FILE." I=3 OPEN"I",#1,A\$+RIGHT\$(STR\$(I),1)	nombre de fichero un nombre de fichero → FILE-3.---?
(6) OPEN "R",#3, "FILE/4.ACC"	→ FILE/4.ACC

- (4) Cuando se escribe un nombre de fichero, también puede escribirse la especificación de una unidad de disco.

LOAD "B:PRGM-A" (Para especificar que PRGM-A se graba desde la unidad de discos B).

La unidad de discos que se va a utilizar se especifica con A, B, C, ó D seguido por dos puntos (:) como delimitador. La unidad de discos principal del equipo de unidades de disco es la A. Cuando se instalan unidades de discos opcionales, entonces B, C, ó D también son válidos. Si en un nombre de fichero no hay ninguna especificación de la unidad de discos, se asume que es la A.

(5) En las sentencias FILES, LFILES, KILL y SET, el asterisco (*) puede representar todos los nombres de ficheros o nombres de ficheros extensos, y un interrogante (?) cualquiera de los caracteres.

Ejemplo:

FILES"B:.*."	Indica que todos los ficheros del disco están en la unidad B.
FILES"*.BAS"	Indica que todos los ficheros tienen "BAS" en sus nombres de fichero extenso.
KILL "TEST??.*"	Indica que cada fichero cuyo nombre es de 6 caracteres que empiezan con TEST debe borrarse.
SET"*.DAT",W	Indica que la protección de escritura se fija en cada fichero cuyo nombre de fichero extenso es DAT.
KILL "A????.BAS"	Indica que cada fichero cuyo nombre es de 5 caracteres empieza con A y el nombre de fichero extenso sea "BAS" debe borrarse.

8.3 Número de Ficheros

En un programa BASIC pueden abrirse hasta 15 ficheros de datos. Cuando se utiliza un fichero, se guarda automáticamente en memoria un área del buffer para la operación de escritura/lectura al disco.

La medida del área del buffer se hace más grande a medida que el número de ficheros de datos incrementa. El usuario debe especificar con anterioridad el número de ficheros para la inicialización del programa BASIC de manera que él o ella utilice la memoria de manera eficiente. (Ver 1.1).

Dado que el espacio reservado en la memoria intermedia es de 300 bytes para cada fichero abierto, se necesitarán 4.5K bytes en el caso de que el usuario desee disponer de 15 ficheros lo cual influirán en otras áreas de la memoria. Al desarrollar un programa se debe considerar el volumen del programa, así como la cantidad de datos para poder determinar el número de ficheros.

El valor del total de bytes de una memoria que aparece por la pantalla en el momento de la iniciación del programa de BASIC es el espacio disponible, una vez reservada la área para la memoria intermedia.

8.4 Instrucciones para los Ficheros

Los ficheros de discos son:

Listado mediante el uso de la sentencia FILES/LFILES (ver 5.19);

Borrado mediante el uso de la sentencia KILL (ver 5.32); y

Renombrado mediante el uso de la sentencia NAMA (ver 5.44).

Pueden estar proveídos de los dos tipos de atributos -el WRITE PROTECT para imposibilitar la escritura y el SYSTEM para prevenir el listado en contra la instrucción de la sentencia FILES/LFILES.

El sistema no podrá escribir ni borrar, mediante la sentencia FILL, ningún fichero que esté proveído del atributo WRITE PROTECT. Para una información más detallada, consulte los anteriores apartados de "PALABRAS DE INSTRUCCION".

8.5 Ficheros de Programas

Los ficheros de programas sirven para cargar el programa almacenado en el fichero del disco, mediante el uso de la sentencia SAVE (ver 5.64). El fichero del programa pasa a la memoria usando la sentencia LOAD (ver 5.38) y quedará intercalado en el programa ya existente, mediante la sentencia MERGE (ver 5.42). Entonces, se ejecuta mediante el comando RUN (ver 5.63). Cuando se especifican los nombres de los ficheros de los programas en las sentencias SAVE, LOAD, RUN, MERGE o CHAIN o bien se usan para la inicialización del Modo BASIC, "BAS" queda añadido automáticamente como un nombre de fichero adicionado, incluso en el caso de que el usuario no haga constar el nombre específico para este fichero.

8.6 Ficheros de DATOS

Los ficheros de datos, clasificados en dos tipos--los ficheros secuenciales y los aleatorios, son los ficheros de disco en los que se cargan los datos. El término datos concierne a los que el usuario ha creado y escrito en la memoria para el programa BASIC y pueden ser leídos siempre que se desee.

Para poder escribir los datos en el fichero o bien leerlos del fichero, el usuario deberá abrir el fichero correspondiente, mediante la sentencia OPEN (ver 5.48) y cerrarlo mediante la de CLOSE (ver 5.6), una vez se haya usado. Sin embargo, cuando se ejecutan las instrucciones END, LOAD, RESET, SYSTEM, DELETE y MERGE, todos los ficheros abiertos quedan cerrados automáticamente.

8.7 Ficheros Secuenciales

En los ficheros secuenciales, tanto la escritura como la lectura de datos empieza desde el principio y de forma cronológica. No se puede escribir ni leer datos desde la mitad del fichero. Los ficheros secuenciales pueden usarse con el siguiente procedimiento.

(1) Escritura de datos en el fichero

(1) Abrir el fichero en modo "O"

(2) Escribir los datos en el fichero *del disco*

(3) Cerrar el fichero

Cuando ya existen datos hasta la mitad del fichero secuencial y se desea continuar escribiendo, el usuario deberá abrir el fichero mediante OPEN "A", ---y así la escritura de datos siguen las líneas anteriores.

(2) Lectura de datos del fichero

(1) Abrir el fichero en modo "I"

(2) Lee los datos del fichero *del disco*

(3) Cerrar el fichero

Ejemplo:

*Galida de los
ficheros de la
memoria*

```
10 OPEN "O", #1, "A-FILE"
20 INPUT "CODE"; C$
30 IF C$ = "0000" x 90 THEN 90
40 INPUT "QUALITY"; N%
50 INPUT "AMOUNT"; A
60 WRITE #1, C$, N%, A
70 PRINT
80 GOTO 20
90 CLOSE #1
```

*En cada am
el fichero*

```
100 OPEN "I", #2, "A -FILE"
110 IF EOF(2) THEN 150
120 INPUT #2, K$, S%, K
130 PRINT K$, TAB(10); S%; TAB(30); K
140 GOTO 110
160 CLOSE #2
```

OPEN "O"....

PRINT# ...

PRINT# USING....

WRITE#

CLOSE#....

OPEN "I"....

INPUT# ...

LINE INPUT#

CLOSE#....

8.8 Ficheros Aleatorios

Random Files

En los ficheros aleatorios, tanto la escritura como la lectura de datos puede empezar en cualquier punto del fichero. Esta función resulta efectiva para el procesamiento aleatorio de datos. El fichero aleatorio puede usarse siguiendo siguiente este proceso:

- 1 Abrir el fichero en modo "R". Es necesario especificar la longitud del registro. En caso que se omita, se le asignará a un registro el valor de "128 bytes".
- 2 Definir el espacio de la memoria intermedia aleatoria.
- 3 Entrar los datos en la memoria intermedia aleatoria (Los datos numéricos deben ser convertidos a los datos de la cadena por adelantado).
- 4 Escribir los datos del área del buffer al fichero del disco mediante la sentencia PUT. Especifique la posición de escritura con el número del registro.
- 3' Lea los datos del fichero del disco al área del buffer mediante la sentencia GET.
- 4' Los datos numéricos deberán convertirse de la cadena al valor numérico.
- 5 Cierre el fichero.

Se pueden escribir hasta 32767 registros en un fichero aleatorio. Los modos de escritura y lectura no se distinguen por su uso del fichero aleatorio. Un fichero aleatorio puede usarse en ambos modos, si es necesario, después de que se han ejecutado las sentencias OPEN y FIELD.

Ejemplo:

```

10 OPEN "R",#2, "R-FILE",50
20 FIELD#2,2 AS C$, 18 AS N$, 30 AS M$
30 INPUT "CODE";C%
40 IF C% = 0 THEN 500
50 INPUT "NAME" ;NA$
60 INPUT "DATA" ;DA$
70 RSET C$=MKI$(C%)
80 LSET N$=NA$
90 LSET M$=DA$ ...

```

OPEN"R",#1,"R-FILE",40

FIELD#1,8 AS A\$,32 AS B\$
LSET A\$=MKD\$(D)
LSET B\$=X\$
Escribir

PUT#1.N%

Leer
GET#1,M%
D=CVD(A\$)
PRINT M%,B\$,D#
CLOSE#1

```

100 PUT#2, C%
110 GOTO 30
.
.
500 INPUT "CODE";N%:IF N%=9999 THEN 700
530 GET#2, N%
540 PRINT TAB (10);CVI (C$) TAB(20);N$;TAB(50);M$
550 PRINT
560 GOTO 500
.
700 CLOSE#2
.
.

```

8.9 Entrada y Salida de Ficheros e Imágenes de Disco

(1) Ficheros secuenciales

Las sentencias PRINT#, PRINT#USING (ver 5.54) y WRITE# sirven para la operación de escritura del fichero secuencial. Las imágenes de disco de los datos de los ficheros secuenciales dependen de las respectivas sentencias.

Donde A\$="RADIO"
 B\$="CASSETTE" }
 C\$="2 3 4 5"

(1) PRINT#1, A\$;B\$;C\$ Imagen de disco
 (2) WRITE#1, A\$;B\$;C\$ RADIOCASSETTE 1 2 3 4 5 <CR>
 "RADIO", "CASSETTE",
 "1 2 3 4 5"<CR>

Cuando se leen estas imágenes de disco,
 (3) INPUT#1,A\$,B\$,C\$

Entonces, A\$= "1 2 3 4 5"

Los datos escritos en (1) se convierten en un elemento de datos. Por lo que al usar la sentencia PRINT para escribir los datos en el disco, el usuario debe pulsar lo siguiente:

(4) PRINT#1,A\$",";B\$," ,";C\$ RADIO, CASSETTE, 1 2 3 4 5
para que, así, los tres elementos de datos puedan ser leídos mediante la instrucción INPUT#. La sentencia WRITE delimita automáticamente cada elemento de datos mediante comillas (" ") e incluye comas (,) entre los elementos. Cuando se usa la sentencia LINE INPUT#, la linea que empieza por WRITE y termina por CR se lee como un elemento de datos.
O cuando D\$="180 Oizumi-machi"
E\$="Gumma, JAPAN" }

(5) PRINT#1, D\$,E\$
Cuando el usuario escribe: 180 Oizumi-machi Gumma, JAPAN

(6) INPUT#1, X\$, Y\$
Se considera que los datos deben estar separados en X\$ y Y\$, mediante el delimitador (coma), por lo tanto,
X\$="180 Oizumi-machi Gumma"
Y\$="JAPAN"
Para escribir o leer los datos como X\$↔D\$ y Y\$↔E\$, el usuario debe incluir " " y CHR\$(&H22).

(7) PRINT 1, CHR\$(&H22);D\$;CHR\$(&H22) ;,"iES
"180 Oizumi-machi", Gumma, JAPAN
O bien deberá usar la sentencia WRITE#, de forma similar à (2).
La sentencia PRINT# escribe los datos numéricos en el fichero del disco, mientras que la de PRINT hace que aparezcan en la pantalla CRT. Por consiguiente,

A=80.5
B=13.15

(Imagen de Disco)

(8) PRINT#1, A, B → A 80.5~~~~~13.15<CR>
(9) PRINT#1, A; B → A 80.5~13.5~ <CR>

Esto se escribe en el disco. Dado que no se han añadido comas en los números numéricos A y B, no pueden ser leídos como datos y aparece un error. Cuando los datos numéricos se escriben en el fichero secuencial, se debe proceder de la siguiente manera:

(10) PRINT#1, A;" ;B → 80.5^,13.15^ <CR>
(11) PRINT#1, USING "###";A;B → 80.5,13.15, <CR>
(12) WRITE#1, A,B,→^80.5,13.15 <CR>

INPUT# sirve para la anterior imagen de disco ((8)-(12)de la siguiente forma:

*INPUT#1,X,Y
(8), (9) Error de tipo adaptación defectuosa
(10) - (12) X=80.5, Y=13.15
*INPUT#1, X\$,Y\$
(8),(9) Entrada pasada y error
(10) X\$="80.5"
(11) X\$="80.50"
(12) X\$="80.5"
*INPUT#1,X\$
(8) X\$=80.5 13.15 "
(9) X\$="80.513.15 "
(10) X\$="80.5"
(11) X\$="80.50"
(12) X\$="80.5"

(2) Ficheros aleatorios

Dado que el fichero aleatorio define el espacio de la memoria intermedia para que sean las áreas de cadena, a través de la sentencia FIELD, los datos que han de escribirse han de ser de tipo cadena. Lo siguiente está especificado como datos de cadena:

Entero MKI\$(I) (Para ser una cadena de 2 bytes)

Número de lectura MKS\$(R) (Para ser una cadena de 4 bytes)

Número real de precisión doble MKD\$(D)

(Para ser una cadena de 8 bytes)

Estos datos de cadena quedan escritos en la área de la memoria intermedia (buffer), mediante la sentencia LSET/RSET y salen usando la sentencia PUT. Al leerlos, vuelven a convertirse en datos numéricos, mediante el uso de la función CVI (para los enteros), CVS (para los números reales) y CVD (para los números reales de precisión doble). Los datos de cadena no varían y son transferidos a la área de la memoria intermedia para su entrada.

(3) Datos Empaquetados

Cada carácter de los datos de cadena se escribe en un byte, sin embargo y para un uso efectivo del disco, los datos empaquetados tienen 2 caracteres en 1 byte. Sólo pueden ser empaquetados 16 tipos de caracteres -son los números que van del 0 al 9, +, -, .(punto), D, E, y un espacio. Esto significa que para los datos empaquetados sólo se permiten los caracteres numéricos. Los datos empaquetados son creados con las funciones PACK\$ y cuando se leen, quedan almacenados por la función UNPACK\$.

A\$="1234.567" 8 bytes
P\$=PACK\$(A\$) 4 bytes.

Cuando se escribe P\$ en el disco, el número de bytes son la mitad de A\$. Si el usuario desea leer los datos, escriba de la siguiente manera:

A\$=UNPACK\$(P\$)

y los datos quedan almacenados así:

A\$="1234.567"

8.10 Número de Ficheros en un Diskette

El número de ficheros permitido en un diskette, incluyendo los ficheros del programa del sistema, cuyo nombre de fichero tiene la extensión ".COM", fichero del programa SBASICII, y todos los ficheros de datos, es de sesenta y cuatro (64).

Apéndice A: Tabla de Códigos de Caracteres

0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	!	,	Ø	@	P	-	p	—		ø	—	—	—	—	—	—
2	"	2	B	R	b	r	B	R		“	”	”	”	”	”	”
3	#	3	C	S	c	s	C	S		’	’	’	’	’	’	’
4	\$	4	D	T	d	t	D	T		£	£	£	£	£	£	£
5	%	5	E	U	e	u	E	U		¤	¤	¤	¤	¤	¤	¤
6	&	6	F	V	f	v	F	V		¤	¤	¤	¤	¤	¤	¤
7	,	7	G	W	g	w	G	W		○	○	○	○	○	○	○
8	(8	H	X	h	x	H	X		□	□	□	□	□	□	□
9)	9	I	Y	i	y	I	Y		△	△	△	△	△	△	△
A	*	:	J	Z	j	z	J	Z		▲	▲	▲	▲	▲	▲	▲
B	+	;	K	[k]	+	;		■	■	■	■	■	■	■
C	,	<	L	\	l	/	,	<		□	□	□	□	□	□	□
D	—	=	M	J	m	j	—	=		△	△	△	△	△	△	△
E	•	>	N	~	n	~	•	>		○	○	○	○	○	○	○
F	/	?	O	—	o	—	/	?		¤	¤	¤	¤	¤	¤	¤

Nota: 1. Parte Superior..... Carácter mostrado
 Parte inferior Código generado mediante combinaciones de tecla

Apéndice B: Mensajes de Error

Errores en hora de edición (Error de edición)

Una vez se ha generado y escrito el programa, se lleva a cabo una simple revisión para determinar si el programa puede ser cargado en la memoria. Las revisiones detalladas, en lo que respecta a la sintaxis, se hacen en el momento de la ejecución.

Estos son los elementos examinados en el momento de la edición:

Se han usado los datos de cadena de forma correcta?

El número de líneas no sobrepasa las 65535?

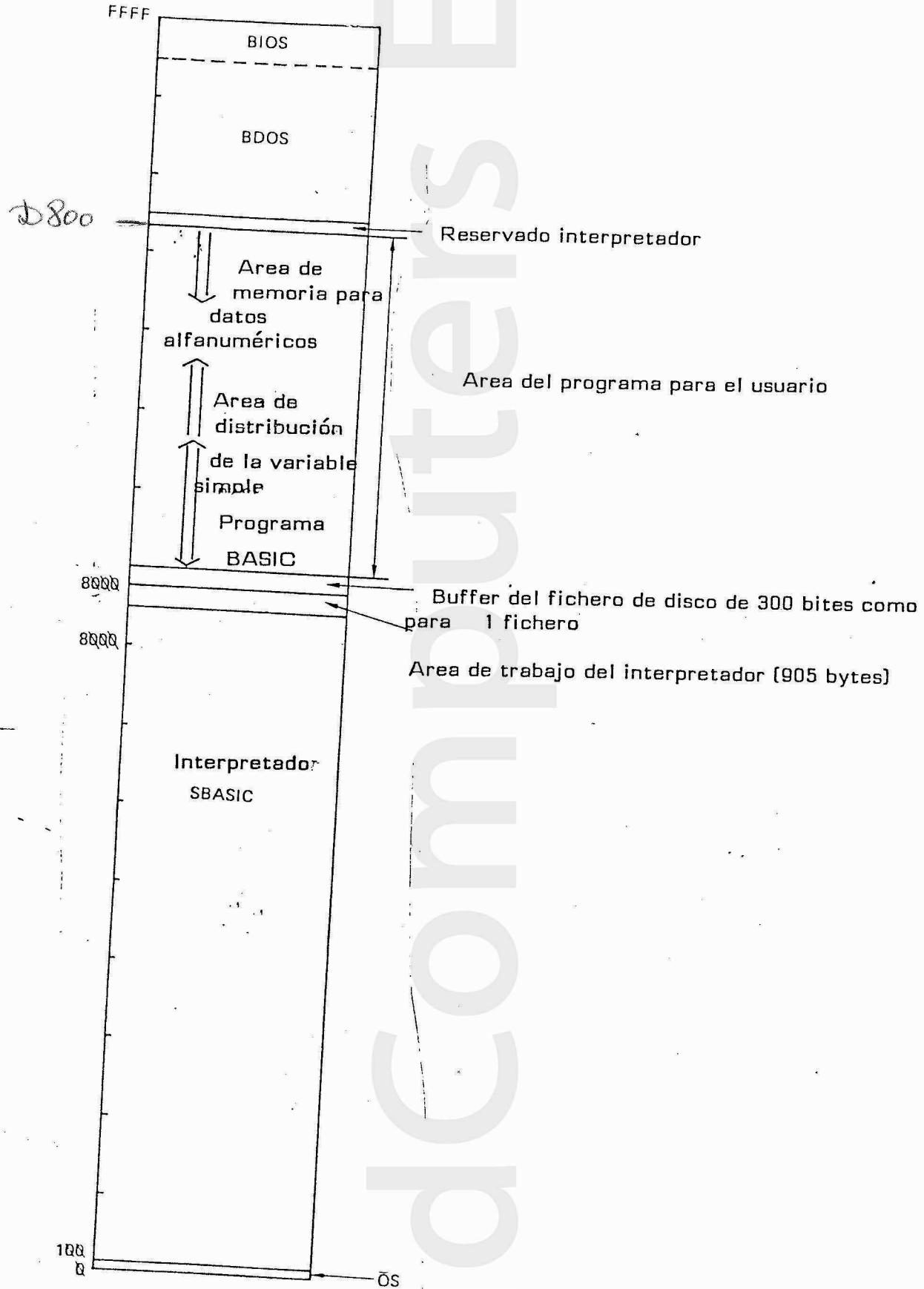
Una vez escrito el programa, está listo para ser cargado en la memoria?

Errores en el momento de la ejecución (Error de Ejecución)

Otros posibles errores que no sean los que se acaban de mencionar, son revisados en el mismo momento de la ejecución. Cuando se encuentra un error, el sistema hace aparecer por la pantalla, el correspondiente mensaje de error y hace volver el control al Modo BASIC. Cuando, en el programa, existe la posibilidad de un error en la rutina, el control pasa a la subrutina de manejo de errores.

Más adelante, veremos los mensajes de error que aparecen por la pantalla y que son revisados en el momento de la edición/ejecución.

Apéndice C: Mapa de la Memoria



LISTA DE MENSAJES DE ERRORES BASICOS

Nº	CODIGO	MENSAJE
1	01	NEXT sin FOR
2	02	Error de sintaxis
3	03	RETURN sin GOSUB
4	04	Fuera de DATA <i>out of DATA</i>
5	05	Llamada de función
6	06	Desbordamiento
7	07	No existe en la memoria
8	08	Número de linea no especificado
9	09	Subíndice fuera del rango (<i>Subscript out of range</i>)
10	0A	Definición duplicada
11	0B	División por cero
12	0C	Directo ilegal
13	0D	Error de escritura <i>(Type mismatch)</i>
14	0E	Fuera del espacio de la cadena alfanumérica
15	0F	Cadena alfanúmerica demasiado larga
16	11	No puede continuar
17	12	Función del usuario indefinida
18	13	Ningún RESUME
19	14	RESUME sin error
20	15	Error que no se puede imprimir
21	18	Posición que no está en pantalla
24	19	Sobrecarga en el campo de impresión
25	1A	FOR sin NEXT
26	1D	WHILE sin WEND
29		
30	1E	WEND sin WHILE

ERRORES DEL DISCO

50	32	Sobrecarga del campo
52	34	Número de fichero erróneo
53	35	No se encuentra el fichero
54	36	Modo de fichero erróneo
55	37	Fichero ya abierto
56	38	Fichero de datos erróneo
57	39	Error de E/S (Entrada/Salida) del disco
58	3A	Fichero ya existente
61	3D	Disco lleno
62	3E	Fin de la última entrada
63	3F	Número de registro erróneo
64	40	Nombre del fichero erróneo
67	43	Demasiados ficheros
68	44	Escritura protegida del fichero
69	45	NAME a través del disco

Lista de palabras de instrucción mediante Función

1. Edición/Ejecución

AUTO	18
CONT	23
DELETE	29
EDIT	31
FILES/LFILES	35
LIST/LLIST	49
LOAD	51
MERGE	53
NEW	55
RENUM	68
RUN	70
SAVE	71
SYSTEM	73
TRON/TROFF	74

2. Sentencias de Declaración

CLEAR	22
COMMON	20
DATA	65
DEF FN	24
DEF USR	25
DEFINT/SNG/DBL/STR	25
DIM	29
ERASE	32
ERROR	32
OPTION BASE	58
REM	67

3. Sentencias de Asignación

LET	46
MID\$	54
SWAP	72

4. Sentencias de Control

CALL	19
CHAIN	20
END	31
FOR/NEXT	35
GOSUB	37
GOTO	38
IF/THEN/ELSE	39
ON ERROR GOTO	55
ON GOSUB	56
ON GOTO	56
RANDOMIZE	65
RESUME	69
STOP	72
WHILE/WEND	75

5. Tecla de Función

KEY	44
KEY LIST/KEY LLIST	45
KEY LOAD/KEY SAVE	45

6. Sentencias de Entrada/Salida de Datos

(Relacionadas con el CRT)

CLS	23
LOCATE	51
PRINT	60
PRINT USING	61
WRITE	76

(Relacionadas con la impresora)

LPRINT/LPRINT USING	52
WIDTH	76

(Relacionadas con el teclado)

INPUT	42
LINE INPUT	47
TINPUT	73

(Relacionadas con el disco)

CLOSE	22
FIELD	34
GET	37
INPUT#	43
KILL	46
LINE INPUT#	48
LSET/RSET	52
NAME	54
OPEN	57
PRINT#/PRINT#USING	64
PUT	64
RESET	69
SET	71
WRITE#	77

(RS-232C, Opcional)

INIT%	40
INPUT%	44
LINE INPUT%	48
PRINT%PRINT%USING	64

(Otros)

BEEP	19
OUT	59
POKE	59
READ/RESTORE	65
WAIT	74

Lista de Funciones

1. Numéricas

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

2. Cadena

- ASC
- CHR\$
- HEX\$
- INSTR
- LEFT\$
- LEN
- MID\$
- OCT\$
- RIGHT\$
- SPACE\$
- STR\$
- STRING\$
- VAL

3. Entrada/Salida

- CSRLIN
- INKEY\$
- INP
- INPUT\$
- LPOS
- POS
- SPC
- TAB

4. Fichero

- ATTR\$
- CVD
- CVI
- CVS
- DSKF
- EOF
- LOC
- LOF
- MKD\$
- MKI\$
- MKS\$
- PACK\$
- UNPACK\$

5. Otros

- ERL
- ERR
- FRE
- PEEK
- USR
- VARPTR

LISTA DE PALABRAS RESERVADAS

ABS	ERASE	LOF	RUN
ALL	ERL	LOG	
AND	ERR	LPOS	SAVE
AS	ERROR	LPRINT	SET
ASC	EXP	LSET	SGN
ATN			SIN
ATTR\$	FIELD	MERGE	SPACE\$
AUTO	FILES	MID\$	SPC
	FIX	MKD\$	SQR
BASE	FN	MKI\$	STEP
BEEP	FOR	MKSS\$	STOP
	FRE	MOD	STR\$
CALL			STRING\$
CDBL	GET	NAME	SUB
CHAIN	GO	NEW	SWAP
CHR\$	GOSUB	NEXT	SYSTEM
CINT	GOTO	NOT	
CLEAR			TAB
CLOSE	HEX\$		TAN
CLS			THEN
COMMON	IF	ON	TINPUT
CONT	IMP	OPEN	TO
COS	INIT	OPTION	TROFF
CSNG	INKEY\$	OR	TRON
CSRLIN	INP	OUT	
CVD	INPUT		UNPACK\$
CVI	INPUT\$		USING
CVS	INSTR		USR
	INT		
DATA			VAL
DEF	KEY		VARPTR
DEFDBL	KILL		
DEFINT			WAIT
DEFSNG	LEFT\$	RANDOMIZE	WEND
DEFSTR	LÉN	READ	WHILE
DELETE	LET	REM	WIDTH
DIM	LFILES	RENUM	WRITE
DSKF	LINE	RESET	
	LIST	RESTORE	XOR
ELSE	LLIST	RESUME	
EDIT	LOAD	RETURN	
END	LOC	RIGHT\$	
EOF	LOCATE	RND	
EQV		RSET	



SANYO

Las especificaciones e información están sujetas a cambios sin previo aviso.