LA ZONA DE VARIABLES (I)

Rafael PAREDES

El Spectrum debe conocer en todo momento las variables que han sido asignadas en un programa BASIC y sus contenidos, de esta manera podrá realizar con exactitud los cálculos y tareas encomendadas. Para almacenar todos los datos, relativos a las variables, utiliza un área de la memoria conocida como: *Zona de variables*, que es de lo que vamos a tratar en este primer capítulo.

La zona de variables no tiene una dirección de comienzo ni longitud, ya que depende de la cantidad de memoria que ocupe el programa BASIC y de la cantidad y tipo de variables asignadas; por tanto, cada vez que se modifica el programa, añadiendo o borrando líneas, el S.O. (sistema operativo) desplaza la información de este área hacia adelante o hacia atrás, dentro del conocido mapa de memoria.

También, cada vez que a una variable se le asigna un nuevo valor, bien sea por un comando directo o como resultado de la ejecución del programa, el S.O. actualiza el contenido de las variables afectadas.

¿Pero cómo identifica el S.O., dentro de la jungla de unos y ceros que es la memoria, los nombres de las variables y sus contenidos? Lógicamente estructurando la información de una forma adecuada, para que con sencillos *algoritmos* sea capaz de interpretarla posteriormente.

TABLA I
CODIFICACION DE LAS VARIABLES

CODIGO		
BINARIO	DECIMAL	TIPO VARIABLE
Ø1Ø	64	Cadena de caracteres
Ø11	96	Numérica (una letra)
1ØØ	128	Matriz numérica
1Ø1	16Ø	Numérica (varias letras)
11Ø	192	Matriz de caracteres
111	224	Control de bucle (FOR/NEXT)

Ubicación relativa

La zona de variables se encuentra ubicada entre la zona donde se almacena el programa BASIC y el área de edición. Para conocer su dirección de comienzo, existe una *variable del sistema*, identificada por el nemotécnico «VARS», cuyo contenido nos lo indica. Esta variable ocupa dos bytes de memoria cuyas direcciones son:

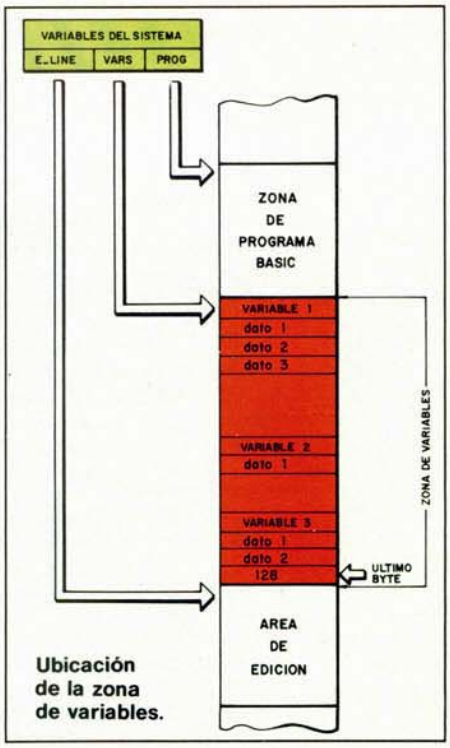
BYTE	DIRECCION
Bajo	23627
Alto	23628

Su contenido puede obtenerse a partir de la expresión:

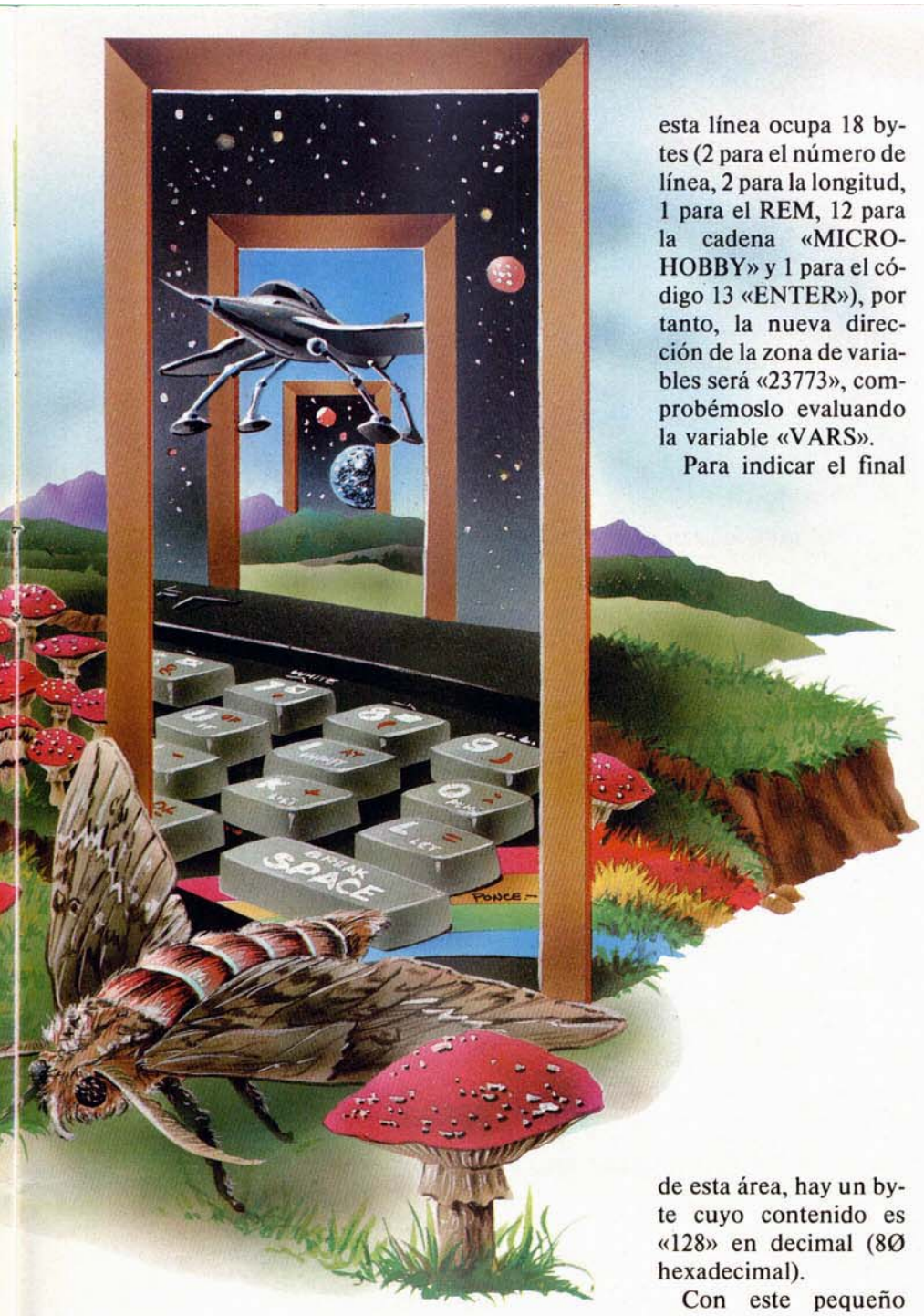
```
PRINT PEEK 23627 + PEEK 23628 • 256
```

Al conectar el Spectrum tiene el valor:

```
23755
```



Ubicación de la zona de variables.



aunque puede ser diferente si tenemos conectado el interface-1 ó el interface de disco, por ejemplo. Este valor coincide con el comienzo de la zona reservada para el programa BASIC (contenido de la variable «PROG»), ¿os sorprende? La explicación es sencilla, ya que debemos tener en cuenta que en un principio la zona de programación no tiene ningún byte reservado; según se van introduciendo líneas de programa, esta zona se va desplazando.

Tecleemos la siguiente línea:

```
10 REM "MICROHOBBY"
```

esta línea ocupa 18 bytes (2 para el número de línea, 2 para la longitud, 1 para el REM, 12 para la cadena «MICROHOBBY» y 1 para el código 13 «ENTER»), por tanto, la nueva dirección de la zona de variables será «23773», comprobémoslo evaluando la variable «VARS».

Para indicar el final de esta área, hay un byte cuyo contenido es «128» en decimal (8Ø hexadecimal).

Con este pequeño programa vamos a «engañar» al S.O., haciéndole creer que no existe la variable «a$» asignada en la línea 1Ø.

```
10 LET a$="MICROMANIA"
20 POKE (PEEK 23627+PEEK 23628
*256),128
30 PRINT a$
```

Esto es debido a que en la línea 2Ø, la sentencia «POKE» almacena el código «128» en el comienzo de la zona de variables, sin reconocer el S.O. ninguna de las variables que estén asignadas, y visualizando, pòr tanto, el conocido mensaje de error:

```
2 Variable not found
```

Con este otro, podremos visualizar la zona de variables y su contenido. Nos servirá para comprender, con mayor facilidad, la forma en que está estructurada esta área.

```
10 REM VARIABLES
20 CLS
30 LET vars=PEEK 23627+PEEK 23
628*256
40 PRINT vars;"    ";PEEK vars,
50 IF PEEK vars>32 THEN PRINT
CHR$ (PEEK vars): GO TO 70
60 PRINT
70 LET vars=vars+1
80 IF vars=PEEK 23641+PEEK 236
42*256 THEN STOP
90 GO TO 40
```

ATENCION

No lo ejecutaremos con la sentencia «RUN» ya que borraría todas las variables asignadas con anterioridad, por el contrario, lo haremos con el comando directo «GO TO 1Ø».

Codificación

Las variables se van almacenando una detrás de otra, de una forma estructurada y codificada. En general, cada variable consta de un primer byte que indica su nombre y el tipo (numérica, de cadena, matriz, etc.), así como de una cantidad variable de bytes que contienen los datos referentes a los valores asignados, a su longitud, etc.

La estructura general del primer byte es:

VARIABLE							
CODIGO			NOMBRE				
7	6	5	4	3	2	1	Ø

Los cinco bits menos significativos (lsb), codificados en binario, indican la cantidad que es necesario añadir al código decimal «96» (comienzo de las minúsculas) o al «64», en el caso de variables de cadena, para identificar la primera letra del nombre de la variable.

Los tres últimos bits, los más significativos (msb), indican el tipo de variable, de acuerdo con la codificación de la tabla I.

LA ZONA DE VARIABLES (II)

Rafael PRADES

El Spectrum almacena todas las variables con las que opera en un determinado espacio de la memoria. En esta segunda parte del análisis de la Zona de Variables os presentamos el resto de los tipos existentes: de cadena, de control de bucle, de matrices numéricas y de matrices alfanuméricas.

La semana pasada vimos cómo está organizada la Zona de Variables, así como la descripción detallada de las Variables Numéricas. Ahora vamos a tratar el resto de los tipos posibles.

Contamos también con un interesante programa del que, por razones de espacio, publicaremos detalladamente su funcionamiento en el próximo número.

Variables de cadena

El primer byte, de este tipo de variables, tiene la siguiente estructura:

BYTE	
0 1 0	X X X X X

El nombre de la cadena viene determinado por el código completo del byte:

```
nombre=CHR$ (PEEK byte)
```

Los dos siguientes bytes determinan la longitud de la cadena, el primero es que de menos peso y el segundo, lógicamente, el de mayor. Cuando una cadena se define como *vacía* (""), estos dos bytes están a "cero".

El valor de la cadena se encuentra distribuido en los siguientes bytes.

EJEMPLO:

```
LET c$="FIN"
```

67	= "C"
3	longitud
0	3 bytes
70	= "F"
73	= "I"
78	= "N"

Variables de control de bucle FOR/NEXT

Las variables de bucle necesitan más bytes para almacenar la información correspondiente a sus límites y al paso.

La estructura del primer byte es la siguiente:

BYTE	
1 1 1	X X X X X

La obtención del nombre puede conseguirse a partir de la expresión:

```
nombre=CHR$ (PEEK byte-128)
```

Los dieciocho bytes posteriores, tienen la siguiente distribución:

- 5 para el límite inferior.
- 5 para el límite superior.
- 5 para el paso.
- 2 para el n.º de línea.
- 1 para el n.º de sentencia.

En un principio, los cinco bytes destinados para almacenar la información correspondiente al límite inferior, tienen el mismo valor que el asignado en la correspondiente sentencia BASIC; pero según se va incrementando el bucle este valor se va actualizando.

EJEMPLO:

```
30 FOR t=2 TO 10 STEP 4
```

244	−128=116 = "t"
0	
0	valor
x	límite
x	inferior
0	
0	

```
9000 REM LECTURA VARIABLES
9010 CLS
9020 PRINT "     NOMBRE     ","
 DIRECCION     ": PRINT
9030 LET vars=PEEK 23627+PEEK 23
628*256
9040 LET paso=0
9050 LET vars=vars+paso
9060 REM BUCLE DE LECTURA
9070 IF PEEK vars>=65 AND PEEK v
ars<=90 THEN GO TO 9150
9080 IF PEEK vars>=97 AND PEEK v
ars<=122 THEN GO TO 9190
9090 IF PEEK vars>=129 AND PEEK
vars<=154 THEN GO TO 9240
9100 IF PEEK vars>=161 AND PEEK
vars<=186 THEN GO TO 9280
9110 IF PEEK vars>=193 AND PEEK
vars<=218 THEN GO TO 9360
9120 IF PEEK vars>=225 AND PEEK
vars<=250 THEN GO TO 9400
9130 IF PEEK vars=128 THEN PRINT
 : PRINT "          FIN
          ": STOP
9140 PRINT : PRINT "
 ERROR          ": STOP
9150 REM CADENA
9160 PRINT CHR$ (PEEK vars)+"$",
"> ";vars
9170 LET paso=PEEK (vars+1)+PEEK
 (vars+2)*256+3
9180 GO TO 9050
9190 REM NUMERICA SIMPLE
9200 PRINT CHR$ (PEEK vars),"> "
;vars
9220 LET paso=6
9230 GO TO 9050
9240 REM MATRIC NUMERICA
9250 PRINT CHR$ (PEEK vars-32);"
()","> ";vars
9260 LET paso=PEEK (vars+1)+PEEK
 (vars+2)*256+3
9270 GO TO 9050
9280 REM NUMERICA COMPUESTA
9290 PRINT CHR$ (PEEK vars-64);
9300 LET paso=vars
9305 LET vars=vars+1
9310 IF PEEK vars>128 THEN PRINT
 CHR$ (PEEK vars-128),"> ";paso:
 GO TO 9340
9320 PRINT CHR$ (PEEK vars);
9330 GO TO 9305
9340 LET paso=6
9350 GO TO 9050
9360 REM MATRIC CADENA
9370 PRINT CHR$ (PEEK vars-96);"
$()","> ";vars
9380 LET paso=PEEK (vars+1)+PEEK
 (vars+2)*256+3
9390 GO TO 9050
9400 REM BUCLE
9410 PRINT CHR$ (PEEK vars-128);
"#","> ";vars
9420 LET paso=19
9430 GO TO 9050
```

En un principio tendrá el valor 2; al final del bucle, alcanzará el inmediatamente posterior al límite, 102 en nuestro caso.

Ø	
Ø	límite
1ØØ	superior
Ø	(100)
Ø	

Ø	
Ø	
4	paso
Ø	(4)
Ø	

44	n.º línea
1	44+1•256=3ØØ

2	sentencia (2–1=1)

Variables de matrices numéricas

El byte primero de una variable destinada para almacenar los datos de una matriz o *array* numérica, tiene la siguiente estructura:

BYTE	
1 Ø Ø	X X X X X

La siguiente expresión permite averiguar su nombre:

```
nombre=CHR$ (PEEK byte–32)
```

Los bytes posteriores tienen la siguiente distribución:

– 2 para la cantidad de bytes ocupados.
– 1 para el número de dimensiones.
– 2 para los valores de cada una de las dimensiones.
– 5 para los valores de cada uno de los elementos.

EJEMPLO:

```
DIM s(2,2)
LET s(1,1)=3
LET s(1,2)=–2Ø
LET s(2,1)= 2
LET s(2,2)=5
```

147	–32=115 = "s"

25	longitud
Ø	25 bytes

2	dimensiones (2)
2	valor 1.ª
Ø	dimensión (2)

2	valor 2.ª
Ø	dimensión (2)

Ø	valor
Ø	primer
3	elemento
Ø	(3)
Ø	

Ø	valor
255	segundo
236	elemento
255	(–2Ø)
Ø	

236+255•256=65516
65516–65536=–2Ø

Ø	valor
Ø	tercer
2	elemento
Ø	(2)
Ø	

Ø	valor
Ø	cuarto
5	elemento
Ø	(5)
Ø	

Variables de matrices de cadena

Las variables de matriz de cadena se almacenan de forma similar a las numéricas. Cuando se define una matriz de este tipo, el área correspondiente a los elementos se rellena con el código decimal "32" (espacio).

El primer byte tiene la siguiente estructura:

BYTE	
1 1 Ø	X X X X X

Para conocer su nombre, utilizar la siguiente expresión:

```
nombre=CHR$ (PEEK byte–96)
```

EJEMPLO:

```
DIM d$(4,5)
LET d$(1)="pepe"
LET d$(2)="juan"
```

196	–96=1ØØ = "d"

25	longitud
Ø	25 bytes

2	bidimensional

4	valor 1.ª
Ø	dimensión (4)

5	valor 2.ª
Ø	dimensión (5)

112	= "p"
1Ø1	= "e" elemento
112	= "p"
1Ø1	= "e" primero
32	= " "

1Ø6	= "j"
117	= "u" elemento
97	= "a"
11Ø	= "n" segundo
32	= " "

32	= " "
32	= " " elemento
32	= " "
32	= " " tercero
32	= " "

Y así sucesivamente.

LA ZONA DE VARIABLES (y III)

Rafael PRADES

Concluimos esta serie sobre la Zona de Variables con un programa de utilidad en la depuración y conocimiento de las variables y las direcciones donde están almacenadas, así como algunos ejemplos de utilización práctica.

El programa número 1, cuyo listado se publicó por razones de espacio en el número anterior, es un complemento de todo lo explicado sobre las variables, a parte de su utilidad en la depuración de programas, ya que permite conocer todas las variables definidas, así como las direcciones a partir de la cual están almacenadas.

Para ejecutarlo utilizar la sentencia:

```
GO TO 9000
```

Las matrices van seguidas de los paréntesis ( ), que permiten una rápida y sencilla localización, y las variables de control de bucle del símbolo #.

Aplicación

Una vez conocidos los códigos con los que se almacenan las variables y la forma en que está estructurada dicha zona, podemos resolver un problema que se plantea al grabar, con la sentencia «SAVE», una cadena de caracteres como matriz numérica; es decir:

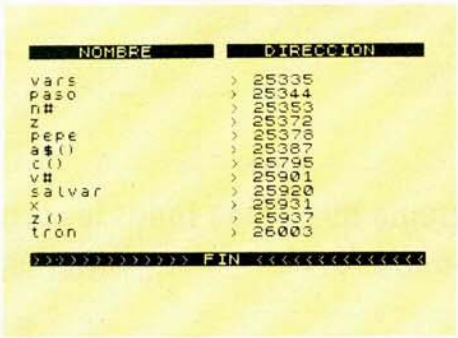
```
LET a$="juanito"
SAVE "nombre" DATA a$( )
```

El S.O. permite que se realice el almacenamiento de esta variable como matriz, aunque a$ no esté dimensionada como tal.

Si inicializamos el ordenador mediante un reset, NEW, RANDOMIZE USR Ø o una nueva conexión, podremos cargar de nuevo los datos almacenados, asignándolos a la misma variable o a otra distinta, con el comando:

```
LOAD "nombre" DATA a$ ( )
```

Pero cuál será nuestra sorpresa cuando no podamos ni visualizar, ni fragmentar, ni asignar un nuevo valor a la variable, ya que el S.O. nos muestra



Ejemplo de utilización del Programa 1.

el siguiente mensaje de error.

```
3 Subscript wrong
```

Analizando detenidamente la memoria con el programa que nos visualiza los contenidos de cada dirección, observamos que ha sido almacenada de la siguiente manera:

193	–96 = 97 = "a"
7	longitud
0	7 bytes
106	= "j"
117	= "u"
97	= "a"
110	= "n"
105	= "i"
116	= "t"
111	= "o"

Los tres bits más significativos del primer byte forman el código de una matriz alfanumérica (11Ø en binario) y los restantes bytes tienen la estructura de una cadena de caracteres. ¿Cómo solucionar esta incompatibilidad entre código y estructura? La forma más sencilla es la de cambiar el código del primer byte para adaptarlo a cadena de caracteres, ya que únicamente se diferencian en el bit de mayor peso (11ØXXXXX-Ø1ØXXXXX).

Para realizar este cambio, podemos utilizar dos métodos; el primero consiste en realizarlo mediante comandos directos, es decir, localizar con ayuda del programa «1», publicado en el número anterior, la dirección de comienzo de la variable y luego ejecutar

```
POKE dirección, PEEK
dirección-128
```

El segundo método, consiste en cambiarlo con ayuda de un pequeño programa en código máquina que se almacena en el buffer de impresión. El programa número «2» se encarga de almacenar el código objeto en dicha área. Puede utilizarlo en cualquiera de sus programas, teniendo en cuenta que su ejecución se realiza con la sentencia:

```
RANDOMIZE USR 23296
```

La explicación de este programa es sencilla:

En el par de registros «HL» se carga la dirección de comienzo de la zona de variables, almacenadas en la dirección «23627» decimal o «5C48» hexadecimal (variable del sistema «VARS»).

En el acumulador (registro «A») se

PROGRAMA 2

```
10 REM CARGADOR A$
20 FOR n=23296 TO 23312
30 READ dato
40 POKE n,dato
50 NEXT n
60 DATA 42,75,92,126,254,193,4
0,6,205,184,25,235,24,245,54,65,
201
```

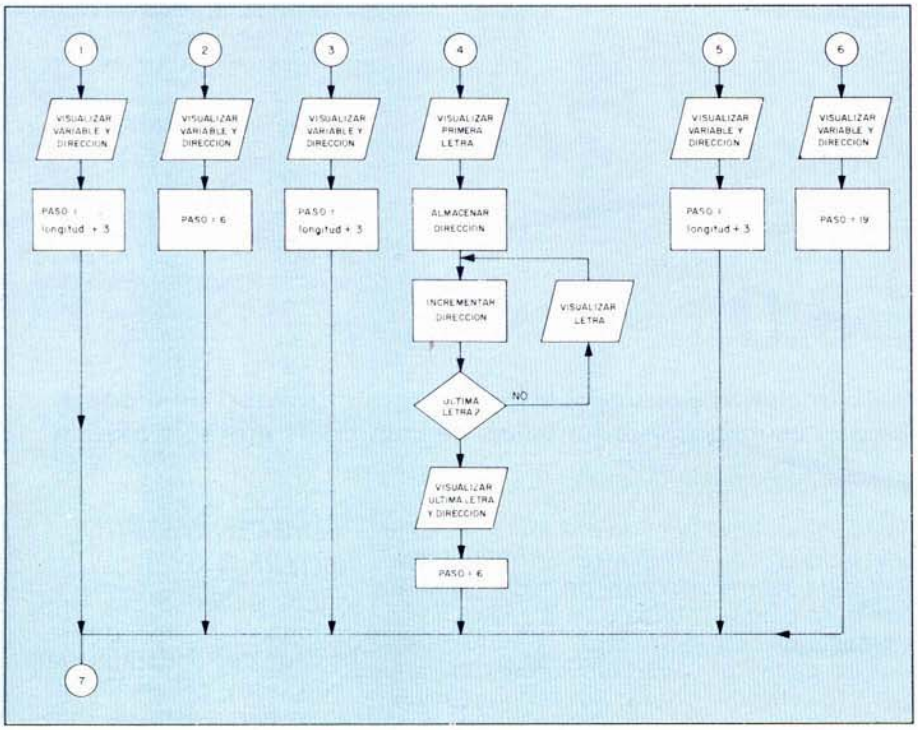


Fig. 3. Pograma 1. Visualización de variables y direcciones.

```
DIRECCION  COD. OBJETO     CODIGO FUENTE
=========  ===========  ======================

  5B00       2A4B5C             LD    HL,(#5C4B)
  5B03       7E         START   LD    A,(HL)
  5B04       FEC1               CP    #C1
  5B06       2806               JR    Z,FIN
  5B08       CDB819             CALL  #19B8
  5B0B       EB                 EX    DE,HL
  5B0C       18F5               JR    START
  5B0E       3641       FIN     LD    (HL),#41
  5B10       C9                 RET
=============================================
```

Listado Assembler del programa "2" con direcciones y datos en hexadecimal.

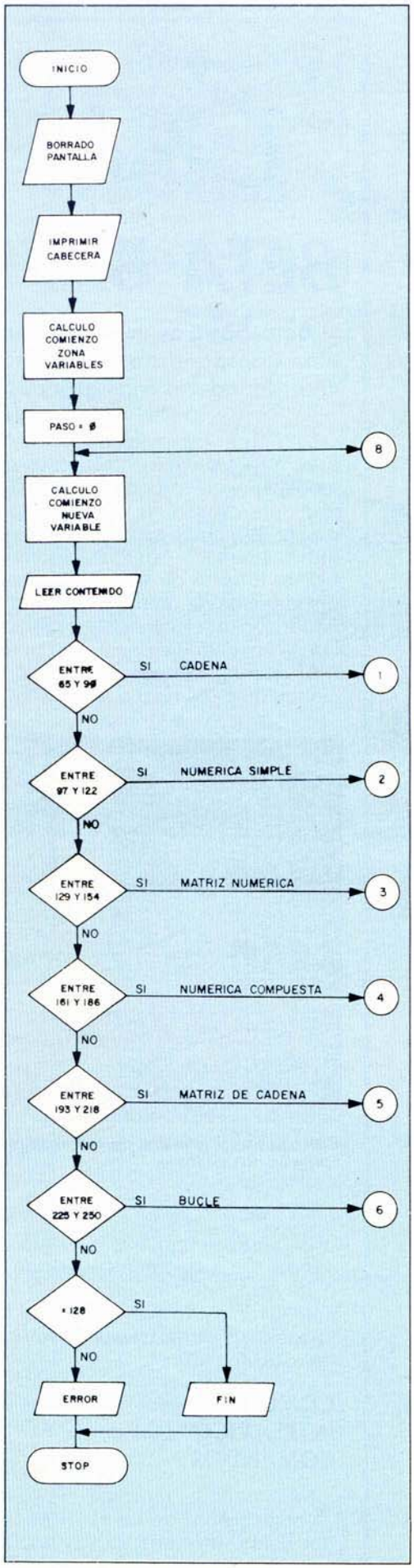
carga el contenido de la dirección almacenada en «HL».

Posteriormente se compara con el valor de la variable a buscar, si es a$ será «193» decimal o «C1» hexa. Si coinciden, se almacena en la dirección apuntada por «HL» el valor correspondiente al código de cadena de caracteres («65» en decimal o «41» en hexa. para la variable a$).

En caso contrario, llama a la rutina de la ROM «NEXT-ONE», ubicada en la dirección «5Ø48» dec. o «19B8» hexa. Básicamente esta rutina calcula el comienzo de una nueva variable, si en «HL» hay una dirección perteneciente a esta zona. El resultado lo devuelve en el par de registros «DE», por tanto será necesario efectuar un intercambio con «HL», para volver a cargar en el acumulador el código de la siguiente variable y efectuar una nueva comparación.

Al final, si la variable es encontrada, el programa retorna al BASIC.

Si en lugar de a$ se utiliza otra variable, será necesario modificar el programa para alterar los valores de comparación y sustitución, localizados en los datos 6(«193») y 16(«65») de la líneas de DATAS.



Pograma 1. Bucle de lectura.