SUPERTRUCOS

SEGURIDAD EN LAS COPIAS

Para muchos usuarios del microdrive y la interface 1, con frecuencia es un problema tratar de obtener copias de seguridad de sus programas favoritos en cinta de cassette, bien sean de juegos o de utilidad, debido a que estos, normalmente escritos en lenguaje máquina, «pisan» la zona que el sistema operativo del Spectrum reserva para las operaciones que debe realizar con el microdrive.

En efecto, la ROM «fantasma» de la interface 1, cuando entra en servicio reserva un buffer de regular tamaño para acomodar sus propias variables de sistema, y una memoria intermedia pare realizar las operaciones de entrada/salida; cualquier comando de los tipos OPEN, MOVE, VERIFY, LOAD y, para el caso que nos ocupa, SAVE que afecte al microdrive, necesita como mínimo 595 bytes libre a partir de una zona determinada de la memoria.

Si el programa que queremos pasar a microdrive ocupa esta zona de memoria, para solucionarlo debemos conocer tres cosas:

A) Dirección en la que carga el lenguaje máquina del programa.

B) Longitud de dicho código máquina.

C) Dirección a reubicar, la cual elegiremos nosotros.

De cada uno de estos datos deberemos obtener un byte alto y un byte bajo, con el fin de dárselos como parámetros al programita en lenguaje máquina que vamos a indicar ahora, debido a J. Antonio García Boal, y que resolverá el problema. Se obtienen de la manera siguiente:

byte alto = INT (dato/256) (FORMULA 1)

byte bajo = ((dato/256) − byte alto) • 256 (FORMULA 2)

Una vez apuntados los tres datos obtenidos de esta forma, teclearemos el siguiente programa, que no es más que un cargador Basic, como siempre:

```
10 FOR N=60000 TO 60011: READ A: POKE N,A:NEXT N
20 DATA 33, BYTE BAJO DE A), BYTE ALTO DE A)
30 DATA 17, BYTE BAJO DE C), BYTE ALTO DE C)
40 DATA 1, BYTE BAJO DE B), BYTE ALTO DE B), 237, 176, 201
50 LET V = USR 60000
```

en donde las palabras byte alto y byte bajo deben ser sustituidas por los valores correspondientes obtenidos del empleo de las fórmulas 1 y 2.

Vamos a ver todo esto con un ejemplo: supongamos que tenemos un bloque de código máquina cuya dirección original de carga es la 23600 y ocupa 10.000 bytes.

Elegimos, por ejemplo, la dirección 30.000 como dirección de carga de momento y tecleamos LOAD " " CODE 30.000. A continuación, lo salvamos en cartucho con la orden.

SAVE *"M";1;"NOMBRE" CODE 30000,10000

Usando las fórmulas 1 y 2 calcularíamos los valores de los bytes alto y bajo, sustituyéndolos en el programa anterior en las líneas 20-40, las cuales quedarían así:

```
20 DATA 33, 48, 117
30 DATA 17, 48, 92
40 DATA 1, 16, 39, 237, 176, 201
```

y salvamos el programa Basic en cartucho. Lo ejecutamos, y si nuestro programa en máquina, que todavía permanece en la memoria, lo requiere, hacemos RANDOMIZE, USR, DIRECCION DE ARRANQUE.

CARGADOR HEXADECIMAL EN CODIGO MAQUINA

Una vez más el lenguaje máquina viene en nuestro auxilio para implementar una pequeña rutina de gran utilidad para los programadores, y que puede emplearse desde Basic sin ningún problema.

Se trata de representar en hexadecimal cualquier número decimal, de 0 a 65535.

Para ello, como puede observarse en la línea número 20 del programa cargador Basic, introducimos nuestro número decimal en la variable SEED, mediante la instrucción RANDOMIZE.

Luego, basta llamar a la rutina en máquina que hace el trabajo duro e instantáneamente aparecerá el número en la pantalla.

Como casi siempre, proponemos como dirección de ensamblado el buffer de impresora, utilizado en el programa cargador.

Para los que posean un ensamblador o estén interesados en averiguar cómo funciona la rutina, proporcionamos también la rutina escrita en lenguaje ensamblador.

```
10          ORG  23296
20 ;
30          RES  0,(IY+2)
40          LD   DE,(23670)
50          LD   C,D
60          XOR  A
70          CP   D
80          JR   Z,BAJO
90          CALL HEXA
100 BAJO    LD   C,E
110         CALL HEXA
120         LD   A,13
130         RST  16
140         RET
150 HEXA    LD   A,C
160         AND  #F0
170         SRL  A
180         SRL  A
190         SRL  A
200         SRL  A
210         CALL LOW
220         LD   A,C
230         AND  #F
240 LOW     ADD  A,"0"
250         CP   58
260         JR   C,PR
270         ADD  A,7
280 PR      RST  16
290         RET
```

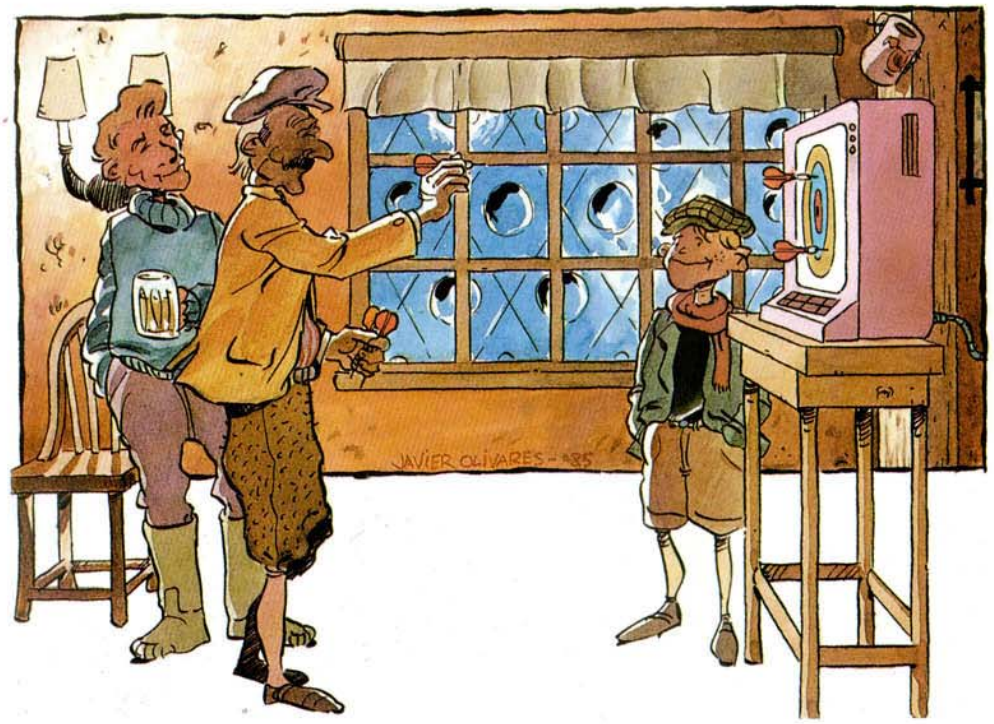
```
  10 INPUT "NUMERO ";CIF
  20 RANDOMIZE CIF: RANDOMIZE USR 23296
  30 GO TO 10
 100 FOR N=23296 TO 23346: READ A: POKE N,A: NEXT N
 110 DATA 253,203,2,134,237,91,118,92,74,175,186,40,3,205,24,91,75,205,24,91,62,13,215,201,121,230,240,203,63,203,63,203,63,203,63,205,41,91,121,230,15,198,48,254,58,56,2,198,7,215,201
```

PARA DIBUJAR COMO QUIERAS

Cambiando los números del FOR, o haciendo operaciones en las coordenadas de los PLOT/DRAW, o bien introduciendo nuevos bucles, conseguiremos infinidad de dibujos, según nos indica E. Sánchez García.

```
 5 FOR a=0 TO 100
10 PLOT 90,a: DRAW 90,a: NEXT
```

INPUT EN CUALQUIER LUGAR DE LA PANTALLA



En algún programa que diseñemos, nos puede interesar, aunque sea por razones puramente estéticas, realizar un INPUT en cualquier parte de la pantalla.

Para poder llevar esto a cabo, nos vemos obligados a volver sobre una variable del sistema que ha aparecido a menudo en esta sección de trucos.

Nos referimos a DEF_SZ, localizada en la posición de memoria 23659.

Como recordaréis, el valor almacenado en esta posición le «dice» a la ROM del Spectrum el número de líneas de la pantalla, comenzando por la parte inferior de la misma, que debe dedicar a la ventana de mensajes y comandos.

En efecto, cuando hacemos un INPUT en medio de un programa, el cursor aparece en las dos líneas inferiores de la pantalla.

Para solucionarlo, en principio la respuesta parece clara: puesto que DEF_SZ almacena el número de líneas de la ventana inferior, pokeemos allí y démosle el valor 24, es decir, toda la pantalla.

Por desgracia, el asunto no resulta tan sencillo. Cuando se ejecuta un comando INPUT, lo primero que hace la rutina de la ROM es volver a asignar a DEF_SZ el valor 2, con lo que nuestro POKE se esfuma de dicha variable y el INPUT se muestra donde siempre.

La solución definitiva está, una vez más, en el lenguaje máquina.

Cuando el ordenador se conecta, el programa de inicialización carga el par de registros IY con la dirección de comienzo del área de variables del sistema. Esto permite al programador acceder a cualquiera de ellas mediante lo que se conoce como «direccionamiento indexado». DEF_SZ se encuentra en IY+49.

Sólo necesitamos dos minúsculas rutinas en ensamblador, una para asignar a DEF_SZ el valor 24 (toda la pantalla, rutina número 1) y la otra para restituir el valor original, 2 (rutina número 2).

Estas dos rutinas son reubicables, esto es, pueden colocarse en cualquier posición de la memoria. Se sugiere el buffer de impresora, pero, por si esa dirección no interesa, aquí tenéis un pequeño cuadro con el código de operación (los números que hay que pokear en la memoria) y los mnemónicos:

RUTINA NUMERO 1

Código de operación
253 54 49 24
2Ø1

Mnemónico
LD (IY+49),24
RET

RUTINA NUMERO 2

Código de operación
253 54 49 2
2Ø1

Mnemónico
LD (IY+49),2
RET

Supongamos que las rutinas 1 y 2 han sido ubicadas en las direcciones «DIR_RUT_2», respectivamente.

Necesitamos ahora un programa Basic que complete a estas dos rutinas, y que las llame en el momento adecuado. Sería el siguiente:

```
1Ø LET ABRE=DIR_RUT_1: LET CIERRA=DIR_RUT_2
2Ø FOR I=ABRE TO ABRE + 4:READ X:POKE I,X: NEXT I
3Ø FOR I=CIERRA TO CIERRA + 4:READ X:POKE I,X:NEXT I
4Ø INPUT "" AND USR ABRE; AT X,Y; N$; "" AND USR CIERRA
5Ø DATA 253,54,49,24,2Ø1
6Ø DATA 253,54,49,2,2Ø1
```

en donde:
la línea 1Ø inicializa las variables ABRE y CIERRA a las direcciones donde se supone que las rutinas 1 y 2 han sido colocadas. Estas direcciones tenéis que decidirlas vosotros y sustituir los números que hayáis escogido por «DIR_RUT_1» y «DIR_RUT_2». Insistiendo en que estos valores pueden ser los que queráis, os sugerimos 23296 para la rutina 1 y 23231 para la rutina 2, con lo que ABRE vale 23296 y CIERRA 23231.

La línea número 4Ø involucra varios trucos:

1. INPUT "" borra las dos líneas inferiores de la pantalla.
2. INPUT "" AND USR ABRE además, llama a la rutina número 1, la cual asigna a DEF_SZ toda la pantalla.
3. X,Y son las coordenadas de la pantalla donde aparecerá el cursor del INPUT, y que tenéis que sustituir por los valores que elijáis.
4. "" AND USR CIERRA llama a la rutina número 2 y devuelve a DEF_SZ su valor original.

Las líneas 2Ø y 3Ø cargan en memoria los bytes de ambas rutinas.

LAS SIETE LLAVES

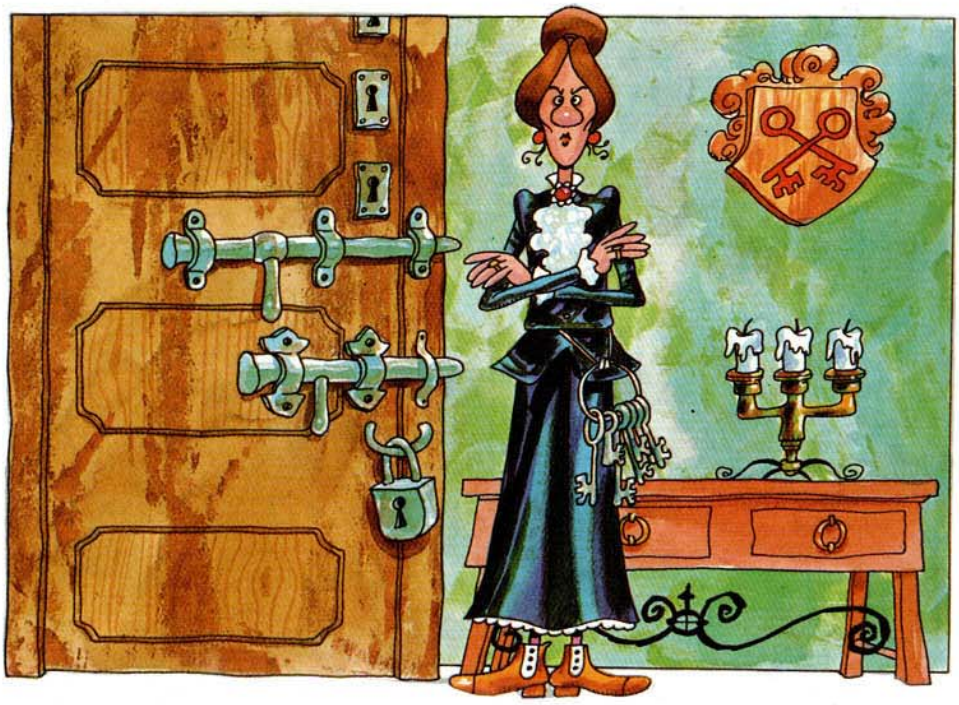
Uno de nuestros lectores de Barcelona, Carlos González, nos manda un auténtico cóctel de trucos para la protección de programas, consiguiendo alejar nuestros listados de miradas curiosas.

El método es el siguiente:

1. encabezar el programa con una línea que diga POKE 23613,Ø
2. continuar con la segunda línea diciendo SAVE «nombre» CODE 23552,long en donde «long» es la longitud de nuestro programa
3. la tercera línea será INPUT «introduce clave»; LINE a$: IF a$ < > «clave» THEN RANDOMIZE USR Ø

Vamos a tratar de explicar lo que sucede en el ordenador cuando estas tres sentencias se ejecutan.

1. La posición de memoria 23613 corresponde a la variable del sistema ERR SP y, en ella, se almacena una dirección a cuyo contenido salta el microprocesador cuando se detecta un error Basic; para verlo con un ejemplo, supongamos que ERR SP contiene el número 31996, el cual hemos averiguado mediante la sentencia PRINT PEEK 23613 + 256*PEEK 23614. Supongamos también que repetimos la misma operación con este número, es decir, PRINT PEEK 31996 + 256*PEEK 3197 y obtenemos 4867. Esta es la dirección a la que saltará el ordenador cuando un error sea detectado. Por tanto, si colocamos en 23613 un cero, estaremos alterando la dirección de salto y, vaya usted a saber dónde irá la CPU cuando la condición de error sea detectada; se producirá el temido «system crash».



2. Lo que efectuamos aquí es grabar en cinta el programa Basic, como si se tratara de Bytes, con la salvedad de que también grabamos las variables del sistema íntegras. Esto implica que, al volver a cargar nuestro programa como LOAD "" CODE, éste se autoejecutará, ya que el valor de algunas de estas variables le impelen a ello.

3. Esta última, requiere menos explicación, ya que el INPUT LINE es conocido por todos nuestros lectores; algunos de ellos nos dirán que de un INPUT LINE se puede salir pulsando CAPS SHIFT + 6 y detener el programa; bien, es cierto a medias, ya que esto provocaría un salto a la famosa ERR SP que hemos alterado previamente; es decir, bloqueo de la máquina garantizado.

CAMBIAR LOS ATRIBUTOS DE LA PANTALLA

Proponemos una corta rutina en código máquina que nos permite cambiar instantáneamente el color del borde, papel y tinta a los valores que elijamos; la rutina se presenta en forma «artesanal», es decir, hay que construir el valor del byte de color y luego introducirlo mediante POKE o bien cambiar el valor en las DATAS.

Si se observa el listado del progama se verá que el número 41 se repite 2 veces; éste es el byte de color que indica papel 5 (cyan) y tinta 1 (blue).

El byte se construye multiplicando el valor del papel por 8 y sumándole la tinta [41=(5•8)+1].

Por ejemplo, para poner papel negro y tinta amarilla, cambiaríamos en las DATAS el valor de 41 por 6[6=(Ø•8)+6].

De paso, hemos incluido unos pocos bytes más que colocan el borde del mismo color del papel, empleando la instrucción RRCA con objeto de mover el número que representa el papel a los bits Ø, 1 y 2 del acumulador. Acto seguido, mediante la instrucción OUT (C), A cuyo equivalente ya vimos en un truco BASIC, conseguimos el efecto deseado.

```
  10 FOR N=40960 TO 40960+22: RE
AD X: POKE N,X: NEXT N
  15 LIST
  20 RANDOMIZE USR 40960
 100 DATA 33,0,88,54,41,17,1,88,
1,255,2,237,176,14,254,62,41,15,
15,15,237,121,201
```

Los colores no quedan fijados de forma permanente, así que si imprimimos algunos caracteres después, sin indicación esplícita de papel y tinta; aparecerán con los atributos que el ordenador conserve por defecto.

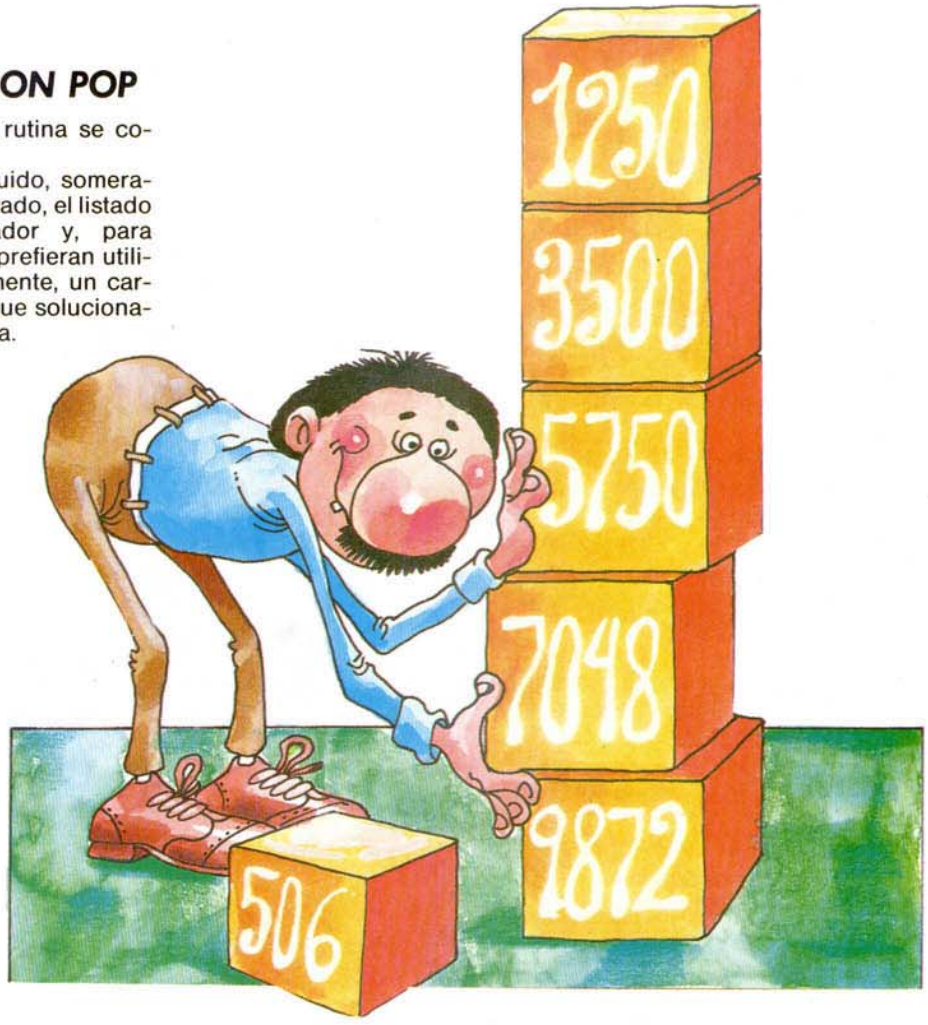
SIMULACION DE LA INSTRUCCION POP

Una de las utilidades del lenguaja Basic que el Spectrum no posee, es la sentencia POP. Este comando tiene la función de impedir que determinada subrutina, que llamemos mediante la sentencia gosub, retorne al programa principal; la utilidad de esta función resulta un tanto oscura y, como siempre, se comprenderá mejor practicándola. Observe que nos permite saltar desde una subrutina a cualquier parte del programa principal.

La rutina se ha escrito en lenguaje máquina y es reubicable, es decir, puede ejecutarse en cualquier parte de la memoria. En principio, se sugiere emplear el buffer de impresora para colocarlo, aunque debe tener en cuenta que si emplea comandos relacionados con la impresora, la rutina se corromperá.

Hemos incluido, someramente comentado, el listado en ensamblador y, para aquellos que prefieran utilizarlo directamente, un cargador Basic que solucionará el programa.

```
10 REM ************************
20 REM    Esta rutina debe
30 REM   colocarse fuera del
40 REM       sistema Basic
50 REM ************************
60 INPUT "Direccion de ensambl
ado? ";dir
70 FOR n=dir TO dir+23: READ a
: POKE n,a: NEXT n
80 DATA 237,123,61,92,59,59,19
3,225,209,122,254,62,202,54,31,5
9,227,235,237,115,61,92,197,201
```



A. PERERA

```
  10 ; SUBRUTINA RELOCALIZABLE
  20 ;
  30         LD   SP,(23613)
  40         DEC  SP
  50         DEC  SP
  60 ;
  70 ; PUNTERO DEL STACK SE LE
  80 ; ASIGNA LA DIRECCION DE
  90 ; RETORNO DE ERROR -2
 100 ;
 110         POP  BC
 120 ;
 130 ; TOMA LA DIRECCION DE
 140 ; RETORNO DE LA ROM QUE
 150 ; TRATA LA SIGUIENTE
 160 ; SENTENCIA BASIC
 170 ;
 180         POP  HL
 190 ;
 200 ; TOMA DORECCION RETORNO
 210 ; DE ERROR
 220 ;
 230         POP  DE
 240 ;
 250 ; TOMA EL ULTIMO DATO DE
 260 ; LA PILA DE GOSUB
 270 ;
 280         LD   A,D
 290         CP   62
 300         JP   Z,#1F36
 310 ;
 320 ; MENSAJE DE "RETURN
 330 ; without GOSUB" SI ES
 340 ; FINAL DE PILA
 350 ;
 360         DEC  SP
 370         EX   (SP),HL
 380         EX   DE,HL
 390         LD   (23613),SP
 400 ;
 410 ; RESTAURA EL PUNTERO DE
 420 ; ERROR
 430 ;
 440         PUSH BC
 450 ;
 460 ; REPONE EN EL STACK LA
 470 ; DIRECCION DE RETORNO DE
 480 ; LA ROM
 490 ;
 500         RET
```

SUPERTRUCOS

SIMULACION DE LA SENTENCIA PRINT USING

Hemos recibido consultas de algunos lectores sobre cómo podrían formatear la salida impresa en la pantalla para conseguir, por ejemplo, una columna alineada de números para programas de aplicación técnica o utilidad.

José María Martínez Arbex nos ha resuelto el problema al enviarnos una pequeña rutina Basic que imita, hasta cierto punto, a la famosa y potente sentencia PRINT USING de otros dialectos de Basic.

El trabajo lo realiza la función definida en la línea 100; necesita dos datos: el número a representar y la longitud del campo donde va a ser representado.

Para flexibilizarla al máximo, hemos definido una variable, LONGCAMPO, inicializada a 15, que nos permite elegir la longitud máxima del campo de representación que queremos.

También está incluida una subrutina para atrapar errores (que el campo sea cero o que su longitud sea menor que la del número a pintar).


```
 10 REM *** SIMULACION DE ***
 20 REM ***  PRINT USING  ***
 30:
 40 REM *** INICIALIZACION ***
 45:
 50   LET SPACE=32: LET E$=""
 60   LET LONGCAMPO=15
 62   LET ERROR=0
 63   LET COMPROBAR=1000
 65:
 70     FOR I=1 TO LONGCAMPO
 80       LET E$=E$+CHR$ (SPACE)
 90     NEXT I
 95:
100   DEF FN U$(N,L)=
       E$( TO L-LEN STR$ N)+
       STR$ N
110:
120 REM *** ENTRADA DATOS ***
125:
130    INPUT "NUMERO ";NUM
140    INPUT "CAMPO ";LCAMPO
145    GO SUB COMPROBAR
147    IF ERROR THEN LET ERROR=
       NOT ERROR: GO TO 130
148:
150    PRINT INVERSE 1;
        FN U$(NUM,LCAMPO)
155:
160  GO TO (130 AND NUM)+
     (2000 AND NOT NUM)
170:
1000 REM *** COMPROBAR ***
1010 REM ***  ERRORES  ***
1020   LET ERROR=NOT LCAMPO OR
            (LEN STR$ NUM>LCAMPO)
1030 RETURN
```

COMO UN PIANO

Con este truco que nos ha mandado José Ignacio Rodríguez Valladolid, podrás convertir tu Spectrum en un elemental piano, donde «Q» será igual a «DO», «2» a «Do» #, «W» a «RE», «3» a «RE» #, «E» a «MI», «R» a «FA», «5» a «FA» #, «T» a «SOL», «6» a «SOL» #, «Y» a «LA», «7» a «LA» #, y «U» a «SI». Se basa en las funciones INKEY$ y BEEP. Para finalizar, apretar el número 1.

```
  10 LET duracion=1: LET tono=0
  20 PAUSE 0
  30 LET tono=(11 AND INKEY$="u"
)+(10 AND INKEY$="7")+(9 AND INK
EY$="y")+(8 AND INKEY$="6")+(7 A
ND INKEY$="t")+(6 AND INKEY$="5"
)+(5 AND INKEY$="r")+(4 AND INKE
Y$="e")+(3 AND INKEY$="3")+(2 AN
D INKEY$="w")+(1 AND INKEY$="2")
+(0 AND INKEY$="q")
  40 IF NOT tono AND INKEY$<>"q"
 THEN STOP
  50 BEEP duracion,tono
  60 GO TO 20
 RETURN L
```

PARA ACENTUAR Y OBTENER LA Ñ

Como habrás «sufrido», por experiencia, la falta de la «Ñ» y de la acentuación en los textos de los programas es un hecho. Por este motivo y para conseguir un perfecto castellano en ellos, José Luis González Sendra nos ha enviado un truco que consiste, precisamente, en esto: añadir los acentos y la ñ, mediante este pequeño listado. Como ha utilizado los GDU, nos manda también sus equivalentes.

```
    1 FOR n=0 TO 47: READ a: POKE
 USR "a"+n,a: NEXT n: DATA 4,8,56
,60,4,60,68,60,0,4,8,56,68,120,64
,60,0,8,16,0,46,16,16,56,0,4,8,56
,68,68,68,56,0,8,16,68,68,68,68,
56,0,56,0,120,68,68,68,68,0
```

A	B	C	D	E	F
á	é	í	ó	ú	ñ

64 COLORES

A pesar de que el Spectrum cuenta con sólo 8 colores disponibles, nada nos impide combinarlos entre si manera que a simple vista dé la sensación de tratarse de un nuevo color. Estos nuevos colores pueden usarse como gráficos, fondos, etc. y para su elaboración se precisan varias fases. Vayamos por partes. En primer lugar hemos de definir una rejilla a la que asignaremos distintos colores de tinta y papel. De este forma la mezcla de colores es más perfecta.

Introduzca las siguientes instrucciones:

Una vez ejecutado el programa, usted probablemente se sorprenderá al ver que todo se borra. Da la impresión de que algo ha fallado. Sin embargo, si el programa estaba correctamente introducido, todo ha ido como se esperaba. Para cerciorarse escriba: PRINT CRH$ 144 seguido de ENTER y verá aparecer en la esquina superior izquierda un pequeño recuadro parecido a un tablero de ajedrez en miniatura.

A continuación, teclee el segundo programa y verá, al hacerlo funcionar, cómo en pantalla le son mostradas las 64 posibles combinaciones de colores, debajo de cada una de las cuales aparecen dos números. El primero se refiere al color del papel, mientras que el segundo es el color de la tinta. Así, si usted desea utilizar el color 75, sólo tiene que escribir PRINT PAPER 7; INK 5; CRH$ 144 e inmediatamente aparecerá un cuadradito del color seleccionado.

Si quiere incorporar este trucod a sus programas, utilizará únicamente el primero, el segundo es sólo una demostración. ¡Ah! y no olvide quitar el NEW de la instrucción 3Ø, pues de lo contrario los efectos serian catastróficos para su recién estrenado programa.

PROGRAMA 1

```
  10 FOR N=0 TO 7: READ X
  20 POKE USR "A"+N,X: NEXT N
  30 NEW
  40 DATA 85,170,85,170,85,170,8
5,170
```

PROGRAMA 2

```
  10 READ  A$
  20 FOR A=0 TO 16 STEP 16
  30 LET D=1
  40 FOR X=1+A TO 15+A STEP 4
  50 FOR N=1 TO 17 STEP 2
  60 LET I=VAL A$(D): LET P=VAL
A$(D+1)
  70 PRINT AT N,X; BRIGHT INT (A
/16); INK I; PAPER P;CHR$ 32+CHR
$ 144
  80 PRINT AT N+1,X;P;I
  90 LET D=D+2
 100 NEXT N: NEXT X: NEXT A
 110 DATA "000102030405060711121
3141516172223242526273334353637
4454647555657666777"
 120 PLOT 128,0: DRAW 0,175
 130 PRINT AT 20,4;"BRILLO 0";AT
 20,20;"BRILLO 1"
```

A LADRON, LADRON Y MEDIO

Joaquín Mateos Lagos nos ha escrito para decir que el truco «Las siete llaves», publicado en el número 16 de nuestra revista, es fácilmente soslayable. La solución que nos da es la de cargar el programa que ha sido salvado en forma de CODE, de esta forma:

CLERA 28999 : LOAD «nombre» CODE 29ØØØ una vez transcurridos unos 15 segundos de la carga, interrumpirla pulsando BREAK y teclear el siguiente programa:

```
  10 FOR n=29000 TO 30000
  20 IF PEEK n>31 THEN PRINT CHR
$ (PEEK n);
  30 IF PEEK n<32 THEN PRINT " "
;
  40 NEXT n
```

Esto permitirá ver claramente cual es la clave que debe ser introducida. A continuación, simplemente cargar bien el programa y, cuando se autoejecute y pida la clave, darle la correcta que hemos anotado anteriormente.

```
 100>FOR N=50000 TO 50053: READ
A: POKE N,A: NEXT N
 110 RANDOMIZE USR 50000
 120 DATA 33,0,64,17,31,64,6,192
,197,213,229,6,15,126,205,122,19
5,79,26,205,122,195,119,121,18,3
5,27,16,240,225,1,32,0,9,209,235
,9,235,193,16,223,201,197,6,8,79
,203,41,203,23,16,250,193,201
```

INVERTIR LA PANTALLA

Una vez más, nos hemos visto obligados a recurrir al código máquina para presentar una utilidad que, sin llegar a ser un largo programa, es bastante más que un truco; así que, como es corto, se puede introducir en cualquier subrutina de su propia aplicación siempre que corra en un Spectrum de 48 K (los poseedores de un 16 K, tendrán que desensamblar el programita y reubicarlo en otras direcciones de memoria).

¿Que qué es lo que hace el programa? pues invierte la pantalla, esto es, nos hace observar todo lo que esté dibujado en ella como si lo miráramos desde dentro del propio televisor; posibles mejoras a esta pequeña subrutina serian aumentar un poco su velocidad, aunque la que posee ahora es bastante aceptable, e invertir también los atributos de la misma forma que hacemos con los caracteres de pantalla.