CAPÍTULO UNDÉCIMO

Código máquina para el Spectrum

PROGRAMANDO EN CÓDIGO MÁQUINA

He mencionado el código máquina una y otra vez, pero ¿qué es exactamente? Recordará que al principio del libro, le mencioné que la parte principal del ordenador (la CPU) sólo entendía variaciones entre alto y bajo voltaje que se podían pensar como dos estados o como unos y ceros. Para comunicarse con la CPU se desarrolló un lenguaje parecido al inglés que se llama BASIC. El programa que reside en la ROM, convierte este lenguaje parecido al inglés en el lenguaje de unos y ceros que es el que entiende la CPU. Como es natural transcurre bastante tiempo mientras el programa que está en la ROM ordena las instrucciones BASIC y las traduce a lenguaje de unos y ceros.

Sin embargo, hay una alternativa que consiste en usar un lenguaje, que está muy cerca del de la máquina y que se llama código máquina. Que no es más que darle a cada uno de los números que componen una instrucción en el lenguaje de unos y ceros, un nombre para que sea más fácil de recordar. A estos nombres se les llama mnemónicos. Usando estos mnemónicos, el lenguaje máquina no está tan lejos del BASIC. Por ejemplo aquí tenemos una instrucción en lenguaje máquina con su equivalente en BASIC:

MNEMÓNICO	LITERAL	BASIC
ld A,34	asigna a A 34	LET A = 34

Como puede ver es cómo escribir abreviaciones en lugar de palabras enteras.

Pero tiene restricciones. En código máquina no se dispone de los lujosos bucles FOR NEXT ni de comandos tan comunes como PRINT. Veamos este ejemplo:

```
200 LET 4 = 23
210 FOR X = 1 TO 100
220 IF X > Y THEN GOTO 20
230 NEXT X
```

Ahora escribiremos el mismo programa sin usar el bucle FOR NEXT. Sería algo así:

```
200 LET X = 1
210 LET Y = 23
220 IF X>Y THEN GOTO 20
230 LET X = X + 1
240 IF X = 100 THEN STOP
250 GOTO 220
```

De este modo el programa es un poco más laborioso. Ahora, para que se parezca más al lenguaje máquina sustituiremos los IF THEN por otro tipo de comparaciones, ya que en lenguaje máquina sólo se puede comparar si una cosa es 0 o no lo es:

```
200 LET X = 0
310 LET B = 23
220 LET A = B
230 LET A = A - 28
240 STOP IF ZERO
250 LET X = X + 1
260 LET A = X
270 LET A = A - 100
280 STOP IF ZERO
290 GOTO 210
```

Aunque el lenguaje usado es distinto, y en lenguaje máquina no existen los números de línea, esta rutina se parece mucho a su equivalente en lenguaje máquina. Como puede ver requiere un poco más de concentración. Ahora veamos cómo sería exactamente nuestro programa en código máquina:

```
999 LD B,100      —Carga el registro B con 100 (LET
                  B = 100)
1000 LD A,B       —Carga el registro A con B (LET
                  A = B)
1001 CP 23        —Compara con 23 (LET A = A —
                  23)
1002 RET Z        RETURN si el resultado es 0
1003 DEC B        Decrementa B (LET B = B — 1)
1004 LD A,B       —Carga el registro A con B (LET
                  A = B)
1005 CP 0         —Compara con 0 (LET A = A — 0)
1006 RET Z        —RETURN si el resultado es 0
1007 JP 1000      —Salta a la línea 1000 (GOTO 1000)
```

Primero una observación: en lenguaje máquina no se utilizan los números de línea, sino que los saltos a una instrucción deben ser enviados a la dirección de memoria en que ésta se encuentra. Aquí he puesto al lado de cada instrucción un número que representa la posición de memoria en que se encuentra, aunque no es un ejemplo muy real ya que las posiciones 999-1007 forman parte de la ROM, y en ellas no es posible colocar ninguna instrucción en código máquina.

Las líneas de las direcciones 1002 y 1006 tienen un RET Z que puede parecerle un poco complicado. Esto simplemente significa RETURN, casi lo mismo que hace un RETURN al final de una subrutina en BASIC. Sólo que aquí el programa posiblemente es llamado desde otro que está en BASIC (y llamar es similar a GOSUB), así que lo que hace esta línea es ver si el último resultado calculado es cero y si esto sucede, vuelve al programa en BASIC para ver lo que sigue.

Se estará preguntando por qué he hecho B igual a 100 al principio y le resto uno cada vez, mirando si es cero, en lugar de incrementarlo para hacer que el programa fuera paralelo con su homólogo en BASIC. Pues bien, lo he hecho para demostrarle que en algunas ocasiones, el lenguaje máquina es más potente que su equivalente en BASIC. A partir de ahora no podrá decir que para hacer la misma operación el lenguaje máquina necesita de tres instrucciones más que su equivalente en BASIC porque las líneas que van de la 1003 a la 1006 ambas inclusive, se pueden reemplazar por la instrucción: DJNZ e

Esta instrucción significa: "Decrementa B y salta 'e' posiciones hacia atrás hasta que el resultado sea 0". En este caso 'e' debe ser un número que haga saltar el programa a la línea 1000.

En esta introducción no he profundizado en absoluto en cuanto a la escritura de código máquina se refiere, pero espero haber dicho lo suficiente para que se sienta tentado de leer algún libro introductorio. De todas maneras voy a comentar otros aspectos. En el ejemplo anterior, he usado números decimales, pero el código máquina se escribe con números **hexadecimales.** La aritmética **hexadecimal** (normalmente llamada hex) es una manera de contar de 16 en 16, en vez de en decenas como es normal, o de 2 en 2 como ocurre en binario. Esto significa que las cifras de un número no tienen estos valores:

$$10^7 \quad 10^6 \quad 10^5 \quad 10^4 \quad 10^3 \quad 10^2 \quad 10^1 \quad 100$$

sino estos:

$$16^7 \quad 16^6 \quad 16^5 \quad 16^4 \quad 16^3 \quad 16^2 \quad 16^1 \quad 160$$

Así, con la primera columna de la derecha se puede contar hasta 15, con la segunda hasta 15, 16, etc. Contar hasta 15 en cada columna es difícil si sólo disponemos de 10 dígitos (del 0 al 9). Por eso en hex, se usan las letras de la A a la F para representar los números del 10 al 15. Así, 0A es 10, 0F es 15, 10 es 16, FF es 255, etc.

Más tarde veremos un programa para invertir los colores de la tinta y del papel en la pantalla.

DÓNDE ALMACENAR EL CÓDIGO MÁQUINA

El programa en código máquina tiene que almacenarse en algún lugar de la memoria. Recuerde que no contiene números de línea, con lo que es muy importante conocer la dirección de cada byte del programa. Si usted ya ha visto algo de lenguaje máquina, quizá con el ZX81, entonces habrá oído decir que es una buena idea colocar el programa en una sentencia REM. Lo que con esto se quiere decir es que primero usted llena 'n' bytes del área de programa (donde 'n' es el número de bytes que ocupa su programa en código máquina) colocando cualquier cosa

detrás de un REM y que estos bytes pueden ser sustituidos luego por instrucciones en código máquina sin afectar para nada al programa en BASIC. En el Spectrum no es necesario colocar el programa en una sentencia REM sino que lo verdaderamente importante es conocer exactamente en qué dirección de memoria empieza el programa. En otros ordenadores, esta dirección es fija (el ZX81, por ejemplo), sin embargo, en el Spectrum puede variar esta dirección según si están conectados dispositivos externos como los microdrives, etc. (vea el mapa de memoria que está en el apéndice). Por suerte, si usted usa REMs para registrar su programa siempre podrá determinar la dirección del comienzo usando la información que nos dan las variables del sistema (vea pág. 174 del manual). Hay dos posiciones, la 23635 y la 23636 que contienen el número que indica la dirección de memoria en que empieza el área de programa. Para hacerlo, use esta fórmula:

```
PRINT PEEK 23635 + 256* PEEK 23636
```

Un sitio aún mejor para almacenar el código máquina es hacerlo por encima de **RAMTOP,** y esto se puede hacer fácilmente usando CLEAR. Por ejemplo CLEAR 30000 limpia la memoria a partir de la posición 30000 y coloca a RAMTOP en dicha posición, con lo que de aquí en adelante todas estas direcciones quedan protegidas de ser borradas con NEW. RAMTOP indica la posición de memoria más alta que puede contener programas BASIC. Esta posición se puede variar para permitir más sitio donde registrar código máquina o datos simplemente. Para editar más fácilmente un programa en código máquina puede ponerlo en líneas DATA. Así, una buena rutina para hacerlo sería:

```
 5 CLEAR 30000
10 RESTORE 200
20 FOR X = 30001 TO 30001 + N (donde W es el
30 READ A                      número de bytes)
40 POKE X,A
50 NEXT X
200 DATA . . . . . .
```

Una vez el programa ya se ha colocado por encima de RAMTOP, puede grabarlo en cinta para recuperarlo cuan-

do lo desee, incluso si hay otro programa en memoria. Para hacerlo utilice los comandos LOAD y SAVE junto con CODE:

SAVE "nombre" CODE 30001,N

En este caso, la primera dirección que se grabará será la 30001 y luego los N bytes siguientes. Para recuperar el programa se hace lo mismo con LOAD.

Para los que deseen trabajar en hex, y almacenar el código máquina en sentencias REM aquí tienen un programa para hacerlo (use letras minúsculas para entrar los códigos):

```
    2 REM XXXXXXXXXXXXXXXXXXXXXXXXXXXX
    3 BORDER 1: INK 7: PAPER 1: CLS
    4 PRINT  INVERSE 1;" Hex-dec converti
dor   o  hex-cargador"'';  INVERSE 1;"Pul
se 'c' para conversion, 'l' para cargar.
        Si pulsa "; FLASH 1;"STOP";
    5 PRINT ; FLASH 0; INVERSE 1;" se det
endra el programa"
    7 LET a=5+(PEEK 23635+256*PEEK 23636)
    8 LET t=0
    9 INPUT q$
   10 IF q$="l" THEN  LET t=1
   20 IF q$="c" THEN  GO TO 50
   25 CLS
   30 IF q$="l" THEN  PRINT  INVERSE 1;"h
ave you set up ..."
   40 INPUT o$: IF o$="s" THEN  STOP
   50 CLS
   60 PRINT "enter m..."
   98 LET x=2
   99 LET y=0
  100 INPUT h$
  105 IF CODE h$>102 OR CODE h$(2)>102 TH
EN  GO TO 240
  110 LET c=CODE h$
  120 IF c>90 THEN  GO TO 220
  130 LET c=c-48
  140 LET i=CODE h$(2)
  150 IF i>90 THEN  GO TO 220
  160 LET i=i-48
```

```
170 LET d=16*c+i
175 PRINT AT 20,0;"
180 PRINT AT x,y;d
181 IF t=1 THEN  GO TO 300
185 LET y=y+4
186 IF y=32 THEN  LET x=x+1
187 IF y=32 THEN  LET y=0
190 GO TO 100
200 LET c=c-87
210 GO TO 140
220 LET i=i-87
230 GO TO 170
240 PRINT AT 20,0;"Mal Hex;vuelva a ent
rar"
250 GO TO 100
300 POKE a,d: LET a=a+1: GO TO 185
```

INVERSIÓN DE LA PANTALLA

Tal como prometí, aquí hay un programa en código máquina que intercambia los colores del papel y de la tinta. Pero no únicamente cambia los colores (que se puede hacer directamente con POKEs en el fichero de atributos) sino que transforma cada carácter en su inverso. La ventaja de esto es que se puede imprimir en la impresora. He puesto esta rutina en la posición 30000, por lo tanto recuerde que antes debe hacer CLEAR 29999

DIRECCIÓN	CÓDIGO	MNEMÓNICO
7530	2AFF3F	LD HL (3FFF)
7533	23	INC HL
7534	7C	LD A,H
7535	FE58	CP 58
7537	C8	RET Z
7538	7E	LD A,(HL)
7539	2F	CPL
753A	77	LD (HL),A
753B	18F6	JR 7533

Esta rutina puede llamarse desde el BASIC escribiendo RAND USR 30000. El usar RAND USR en lugar de PRINT USR tiene la ventaja de que se ejecuta el programa y cuando termina no imprime nada en la pantalla. La forma

en que actúa el programa es muy simple. El registro de 2 bytes llamado HL es cargado con el valor de la dirección en donde empieza el archivo de imagen (es decir, en qué lugar se encuentra la pantalla en memoria), menos uno (ya que enseguida se incrementa). El registro A se carga con el contenido de H, y en la posición 7535 se compueba que no supere el valor 58, ya que si esto ocurre quiere decir que hemos llegado al final del archivo de imagen y se detiene el programa en 7537. Si no se ha llegado al final del archivo de imagen, entonces el contenido de HL está apuntando a una posición de la pantalla cuyo carácter queremos invertir. Esto se hace cargando en A el byte contenido en la dirección indicada por HL (en el mnemónico este hecho se representa colocando HL entre paréntesis) y luego tomando el CoMPLemento de A. El complemento de un código de carácter es justamente su imagen inversa que es lo que queremos. Ahora lo colocamos de nuevo en la pantalla efectuando la operación inversa en 735A. Una vez hecho esto, la próxima instrucción le dice al programa que salte 10 bytes hacia atrás donde HL se incrementa de nuevo y se repite todo el proceso para una nueva zona de la pantalla.

Si como espero aún no ha tenido bastante, aquí hay otra rutina que cambia los colores pero sin cambiar para nada el texto. Esto se puede hacer también usando POKE pero es mucho más lento:

	Decimal Bytes	Mnemónico	Comentario
	6,235	LD B,235	
	22,40	LD, D,40	Código Ink/Paper
	33,0,88	LD HL,22528	Comienzo del archivo de atributos.
dirección 32528	114	LD (HL), D	
	35	INC HL	
	114	LD(HL), D	
	35	INC HL	
	114	LD(HL, D	
	35	INC HL	
	5	DEC B	¿Es el final del archivo?
	194,16,127	JPNZ,32528	
	122	LD A,D	

```
50,141,92  LD(23693), A Hace permanentes
                        los colores
201        RET
```

El código del color y la tinta puede cambiarlo, se encuentra en la segunda línea y yo he puesto 40, pero usted puede poner cualquier otro número. Recuerde que este número es el de ATTR y se calcula como: 8 multiplicado por el código del papel más el código de la tinta. Luego deberá pasarlo a hexadecimal.

CÓMO OBTENER EL MÁXIMO DE MEMORIA

Si usted necesita más sitio en memoria para programas o datos, aquí tiene algunos trucos.

Puede desplazar el comienzo del área de los caracteres definidos por el usuario para obtener más sitio para su programa en BASIC. Esto depende, claro, del número de gráficos definibles que use en este programa. Pero si no usa ninguno, conseguirá 21*8 bytes más. Este lugar, también es bueno para almacenar código máquina. La dirección es siempre conocida ya que tanto en el Spectrum de 16K como en el de 48K, USR "a" nos da el comienzo de esta área. Así un programa para entrar código máquina utilizando esto sería:

```
10 FOR A = USR "a" TO USR "h"
20 READ X: POKE a,X
30 NEXT a
40 DATA . . . . . .
```

Otro lugar en el que puede colocar datos temporalmente es en el área del archivo de imagen. Deje una pequeña ventana en la pantalla para los mensajes, y en el resto coloque tanto la tinta como el papel del mismo color. Debe evitar usar CLS y también escribir en el área reservada pues se perderían los datos.