ediciones
técnicas
REDE
BARCELONA
(ESPAÑA)

código máquina * aplicaciones * spectrum * david laine

SERIE
AUTOAPRENDIZAJE
ACELERADO

colección

código máquina

aplicaciones

(programación avanzada)

para
ZX
spectrum

david laine

SUMARIO

CAPITULO 1: INTRODUCCION 9

CAPITULO 2: LA PROGRAMACION 11
La fiabilidad de un programa. La sencillez de un programa. La facilidad de comprobación de un programa. La velocidad de ejecución de un programa. El tamaño de un programa. La documentación. Los accesorios del programa. Las especificaciones del programa.

CAPITULO 3: LAS INSTRUCCIONES 15
La pila ("stack").

CAPITULO 4: LA REPRESENTACION DE NUMEROS 25
Los números en coma flotante. La multiplicación y la división de números en coma flotante. Las estructuras de datos. La aritmética con y sin signo.

CAPITULO 5: EL DIRECCIONAMIENTO 31
El método directo. El método directo con un desplazamiento fijo. El método indirecto. El direccionamiento indexado. El método indirecto múltiple. El método encadenado. La computación de direcciones (e instrucciones).

CAPITULO 6: LOS PRINCIPIOS ELEMENTALES 39
Introducción. El borrado de la memoria de pantalla. Un ensayo en tiempos de ejecución. La preparación del área de atributos. La rutina de espera (WAIT) y el traspaso de datos entre subrutinas.

CAPITULO 7: LA PRESENTACION EN PANTALLA 47
La colocación de un "pixel" en la memoria de pantalla. La salida. PRIN. PTEX. SRHL. RHL3. PRT8. RPORT. MAP8. Sinopsis.

CAPITULO 8: LA ANIMACION 71
GCELL. La comunicación con el BASIC. Datos de la celda. Descripción de GCELL. Trazado de la celda (de LXX en adelante). SEBIT.

CAPITULO 9: EL TRATAMIENTO DE ERRORES Y EL TRASPASO DE PARAMETROS 83
Tratamiento de los retornos por error. El traspaso de nombres de variables (parámetros). VAR$1, una rutina para la búsqueda de variables en el área de variables. Documentación de la rutina. PCALL. VAR$1. Sinopsis.

CAPITULO 10: RUTINA PARA ORDENAR NUMEROS EN COMA FLOTANTE 105
Una solución que permite realizar en 145 segundos lo que exige cinco horas en BASIC. SORTF. COMPF. Un ejemplo práctico de SORTF.

CAPITULO 11: EL TRASPASO DE OTROS PARAMETROS... 115
OPARS (Otros parámetros). Operación. Números. Fin de parámetro (EOPAR). Inicio de cadenas (STSTR). Sinopsis.

CAPITULO 12: BORRADO DE BLOQUES DE BASIC 129
SUPLN. CNXLN. RESLN. Operación de SUPLN. Operación de BLOCK. BDLE1. BDLE2.

CAPITULO 13: AREA DE ATRIBUTOS..... 137
Operación.

CAPITULO 14: GRAFICOS EN ALTA RESOLUCION..... 143
Operación de PLINE. XLINE. NPOIN. SDIFF. DRAWL (Trazado de una lista de líneas). GVALB. BYTEV: BYTe EValuado. MOVEC. Operación de MOVEC. Otros detalles. DRAWA: Trazado de matriz. Operación de DRAWA. Programa de dibujo en BASIC. Sinopsis.

CAPITULO 15: MISCELANEA 165
BCD o Decimal Codificado Binario. Modificaciones. DEMO1. DEMO2. Entradas múltiples. La acción recursiva o la serpiente que se muerde la cola. Notas sobre el código máquina y el ensamblador. Código máquina — Lo que debe y no debe hacer.

Capitulo 1

INTRODUCCION

Este libro no está dirigido a los principiantes sino a los que ya hayan utilizado programas en código máquina publicados en libros o revistas y sientan la necesidad de profundizar en ello por sus propios medios.

Para aquellos que todavía se muestren interesados, este libro no es una tesis sobre códigos de instrucciones u operaciones internas del Spectrum. Si Vd. no lo tiene aún, deberá conseguir un libro que explique detalladamente las funciones del microprocesador Z 80, algunas de las cuales serán explicadas en el presente libro pero otras serán omitidas a pesar de su familiaridad o simplemente porque no habrá oportunidad de detallarlas. Se incluye un cuadro sinóptico de las instrucciones disponibles y de sus tiempos de ejecución. Sin embargo, no se aborda el tema de la programación de periféricos, ni del registro vector de interrupción, ni tampoco del registro de refresco de memoria. Mi propósito es presentar una introducción a los programas en código máquina que puedan conjugarse con el BASIC, el cual supongo que será ya bien conocido por el lector.

¿Por qué razón debe utilizarse el código máquina?

Para gozar de una libertad total frente a las limitaciones del BASIC y de un notable aumento en la velocidad de ejecución. He incluido una rutina para tratamiento de conjuntos numéricos (Capitulo 10) que es aproximadamente 125 veces más rápida que su equivalente en BASIC (y luego se indica cómo se puede aún doblar la velocidad!). Por otro lado, los errores en código máquina se manifiestan de una forma mucho más rápida.

Para los programas en código máquina, he utilizado un simple ensamblador de Picturesque.

Recuerde siempre que si Vd. puede ver una manera lógica de resolver un problema, esto significa que el problema puede ser resuelto.

Lo más difícil para un principiante es llegar a la solución, perseverar en su búsqueda hasta que el último error haya sido eliminado y el programa funcione correctamente.

No intente, al menos al principio, pasarse más de una o dos horas de un tirón con ello, y procure siempre tomar nota de todos sus errores. Después de unas semanas, los peores momentos de nervios habrán pasado ya y Vd. se habrá familiarizado con el código máquina.

Ante cualquier problema, escriba exactamente lo que quisiera conseguir y después trace los diagramas de flujo. Si no puede resolver una pequeña parte del problema, guárdela de momento en una cajita —es decir, apartada— y continúe con el problema principal. Después vuelva a trabajar con los problemas apartados como si fueran tan importantes como el problema principal.

Finalmente, no olvide esto: el verdadero programador se manifiesta en una de estas dos situaciones: en las profundidades de la desesperación porque el programa no funciona o en exaltado júbilo por conocer la causa que impide que el programa funcione.

Capítulo 2

LA PROGRAMACIÓN

«Un ingeniero fue llamado desde lejos. La máquina no funcionaba. El estudió el problema. Pidió un martillo. Le propinó un formidable golpe a la máquina. Funcionó. Más tarde llegó la factura:

Viaje y transporte	*10.000,–*
Golpear la máquina	*5,–*
Saber qué y dónde golpear	*100.000,–*
TOTAL	*110.005 ptas. (+ I.T.E.)»*

(«Modernised Apocryphal»)

La programación es un arte más que una ciencia. La ciencia, no obstante, también forma parte de ella ya que las reglas impuestas por las instrucciones en código máquina y por cualquier sistema operativo no admiten flexibilidad. Pero la presencia de los mejores ingredientes no implica necesariamente un buen resultado en términos culinarios cuando el cocinero es un gorila, sino que, por el contrario, un gran cocinero sabrá lograr un resultado excelente a partir de los principios menos prometedores.

Hay una constante acción recíproca entre ocho factores:

- Fiabilidad
- Sencillez
- Facilidad de comprobación
- Velocidad
- Tamaño
- Documentación
- Accesorios del programa
- Especificaciones del programa

LA FIABILIDAD DE UN PROGRAMA
Conjetura de Djikstra:

Si un programa tiene n instrucciones, cada una de ellas con una probabilidad p de que cumpla su cometido, la probabilidad de que el programa funcione correctamente es del orden de p^n.

Si el programa forma un bucle con L vueltas, entonces la probabilidad es del orden de p^{nL}, lo cual significa que si p no es igual a uno el programa no merece ser ejecutado.

Cada defecto en un programa deberia ser investigado, explicado y corregido. Un programa defectuoso no merece ser utilizado. Una coma mal situada ha costado a veces millones.

LA SENCILLEZ DE UN PROGRAMA

No tiene ningún mérito realizar programas innecesariamente complicados. Lo más complicado puede siempre ser dividido en unas cuantas partes y éstas, a su vez, pueden ser reducidas a partes más sencillas todavia. Yo creo que una buena forma de comprobar un programa es realizar el listado desde las instrucciones de salto a las etiquetas más adecuadas. Los resultados suelen ser manifiestamente provechosos.

LA FACILIDAD DE COMPROBACION DE UN PROGRAMA

Se ha escrito ya mucho sobre este tema y todo lo que voy a señalar aqui es que la sencillez en la estructura facilita mucho la comprobación de un programa. Puede haber más combinaciones de bits en sólo 40 bytes que átomos en el universo.

LA VELOCIDAD DE EJECUCION DE UN PROGRAMA

Cada instrucción precisa de un tiempo finito para ser ejecutada, y existen siempre varias combinaciones posibles de instrucciones que, mezcladas, pueden producir el mismo resultado. Si observa que una parte de un programa parece especialmente lenta en producir resultados, examinela detenidamente para ver qué bucles hay dentro de otros bucles. El aumento de velocidad puede requerir cambios en la estructura y repercutir, por tanto, en un programa más largo.

EL TAMAÑO DE UN PROGRAMA

«Cualquier persona puede construir un puente pero únicamente un ingeniero puede construir adecuadamente un puente».

El tamaño de un programa es la suma de dos partes: las instrucciones y el conjunto de datos.

Los datos no deberian formar parte de un programa excepto, quizás, en programas de prueba. El programa deberia trabajar con ayuda de un puntero que localizara los datos necesarios en el lugar donde éstos se encontraran y permitirles que cumplieran su misión.

El número de instrucciones puede ser casi siempre reducido. Cuanto más directa sea la construcción del programa, más efectiva y fácil será esta reducción.

LA DOCUMENTACION

Un programa o una subrutina desprovista de una documentación adecuada puede tener muy poca utilidad. Se pueden retener en la memoria los detalles de utilización de un programa durante unos tres meses pero, una vez transcurridos éstos, el programa presentará un riesgo elevado de fallar cada vez que se utilice nuevamente.

La documentación no consiste tampoco en una gran obra, sino que tan sólo es preciso que conste de:

- Lista de las condiciones de entrada
  - a) registros
  - b) posiciones especiales
- Lista de las condiciones de salida
  - a) registros
  - b) posiciones especiales
  - c) registros protegidos
  - d) estado de los señalizadores (banderas o «flags»)
- Breve descripción de la función

Todo esto, junto con un listado y un diagrama de flujo, deberia guardarse en una buena libreta de notas con tapas rigidas. Si puede guardar también el código fuente original en cinta, mejor todavia. Yo suelo guardar mensajes en los mismos casetes y parece que da buen resultado.

LOS ACCESORIOS DEL PROGRAMA

Cuando se habla de lo que se necesita además del programa

para ejecutarlo, sin demasiada fantasía, se puede decir que se trata de los periféricos necesarios además del televisor. Debe adecuarse siempre la presentación al dispositivo de salida que vaya a utilizarse. Lo que podría ser muy impresionante en destellos y color tendrá un aspecto muy diferente en una impresora ZX.

LAS ESPECIFICACIONES DEL PROGRAMA

Este tema se ha dejado para el final porque todo lo demás le afecta y queda afectado por las especificaciones. Deben darse varias vueltas al asunto hasta llegar a un compromiso aceptable.

Algunos aspectos particulares merecen ser destacados si Vd. está preparando un programa para otra persona:

a) ¿Entiende Vd. bien lo que él le dice que necesita?

b) Lo que dice que necesita, ¿es una expresión verdadera de lo que realmente necesita? Recuerde que Vd. y esa persona han de tener una coincidencia de apreciaciones sobre el problema que ha de resolverse.

c) ¿Puede Vd. ver el problema como otro planteamiento general que haya resuelto antes? O, dicho con otras palabras, ¿ha solucionado Vd. algo como esto ya? ¿Será este problema el primero de una serie? ¿Sería más conveniente escribir un programa más general para prever futuras necesidades? Por ejemplo: dada la necesidad de trabajar con aritmética de hasta 7 bytes, ¿no sería mejor prever soluciones generales para N bytes y luego asignar el valor 7 a N para el caso específico?

d) Si el problema es muy amplio el tiempo empleado en diseñar la base de datos puede significar luego un ahorro de tiempo a la hora de manejar los propios datos. Todos los datos que se refieran a un tema concreto deberían almacenarse juntos, para poder acceder a ellos a través de un registro de página. Los diferentes valores dados al registro de página se utilizarán para acceder a los distintos datos.

e) Cuando ya cuente con un esquema de la solución, tendrá también algunas preguntas que hacerse. Por lo tanto, vuelva a a) y comience de nuevo.

Capítulo 3

LAS INSTRUCCIONES

Los códigos de las instrucciones y sus influencias sobre los señalizadores (banderas/«flags») están condensados en las figuras 3.1 y 3.2, junto con las combinaciones de direcciones permitidas. Estas tablas no sustituyen a los libros mencionados en el capítulo 1.

Disposición de la *Figura 3.1:*

columna	*descripción*
1	Mnemónico de la instrucción
2	Operación simbólica
3	Combinaciones de direcciones permitidas (donde están permitidas dos tipos de éstas, los dos grupos posibles se hallan separados por un espacio).

Los números colocados debajo de algunas direcciones indican los tiempos de ejecución de la operación asociada (en ciclos de reloj del ordenador).

N	indica que puede utilizarse un valor de 1 byte.
NN	indica que puede utilizarse un valor de 2 bytes.
(NN)	indica que puede utilizarse la dirección de un byte.
d	es el desplazamiento de 1 byte utilizado con un registro de página.
DISP	es el desplazamiento en el salto hacia una instrucción cercana.

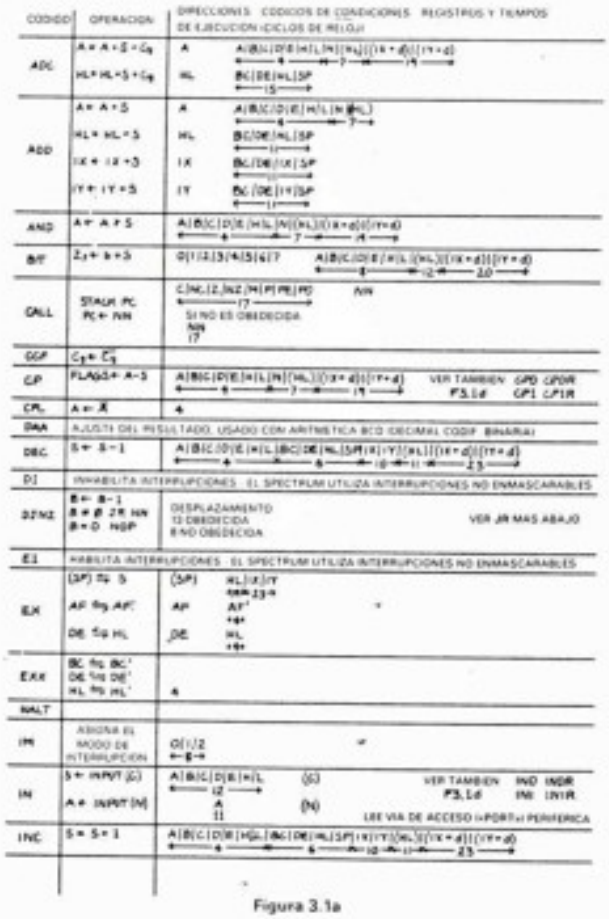


Figura 3.1a

Figura 3.1b

RLC	ROTACION IZQUIERDA	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) ← 8 ← 15 ← 23	VER F3.2 R1
RLCA	ROT. DER. DE A	4	VER F3.2 R2
RLD	ROTACION IZDA. DE A Y (HL)	18 BCD A Y (HL) POR PARTES	VER F3.2 R3
RR	ROTACION DERECHA	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) → 8 → 15 → 23	VER F3.2 R4
RRA	ROT.DER. DE A	4	VER F3.2 R4
RRC	ROTACION DERECHA	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) → 8 → 15 → 23	VER F3.2 R5
RRCA	ROT. DER.	4	VER F3.2 R5
RRD	ROTACION DERECHA DE A Y (HL)	18 BCD A Y (HL) POR PARTES	VER F3.2 R6
RST	PC EN LA PILA PC ← p	01 8|16|24|32|40|48|56 PARA SALTOS AL ENCABEZAMIENTO DE LA ROM ← 11 →	
SBC	A ← A-S-C_y	A	A|B|C|D|E|H|L|N|(HL)|(IX+d)|(IY+d) ← 4 ← 7 ← 19 →
	HL ← HL-S-C_y	HL	BC|DE|HL|SP ← 15 →
SCF	PONE A 1 LA SEÑAL DE C_y	4	
SET	S_b ← 1	0|1|2|3|4|5|6|7	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) ← 8 ← 15 ← 23 →
SLA	DESPLAZAMIEN. IZDA. ARITMETI.	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) ← 8 ← 15 ← 23 →	VER F3.2 S1
SRA	DESPLAZAMIEN. DER.ARITMETICO	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) ← 8 ← 15 ← 23 →	VER F3.2 S2
SRL	DESPLAZAMIEN. DERECHA LOGICO	A|B|C|D|E|H|L|(HL)|(IX+d)|(IY+d) ← 8 ← 15 ← 23 →	VER F3.2 S3
SUB	A ← A-S	A|B|C|D|E|H|L|N|(HL)|(IX+d)|(IY+d) ← 4 ← 7 ← 19 →	
XOR	A ← A ⊕ S	A|B|C|D|E|H|L|N|(HL)|(IX+d)|(IY+d) ← 4 ← 7 ← 19 →	

C_y ACARREO

Figura 3.1c

BLOQUES, REPETICION, CARGA Y COMPARACION

CPD	FLAGS ← A-(HL) HL ← HL-1 BC ← BC-1	COMPARACION CP	TODAS LAS INSTRUCCIONES DE COMPARACION DEJAN INALTERADO EL REGISTRO A. ALTERANDO SIN EMBARGO, LOS BANDERINES SEGUN EL RESULTADO DE LA SUSTRACCION
CPI	HL ← HL+1 BC ← BC-1		
CPIR	HL ← HL-1 BC ← BC-1	REPETICION HASTA QUE A=(HL) O BC=0	D O I SIGNIFICAN DECREMENTAR O INCREMENTAR EL CONTENIDO DE HL EL CONTENIDO DEL REGISTRO DOBLE BC ES DECREMENTADO EN TODOS LOS CASOS
CPDR	HL ← HL+1 BC ← BC-1	REPETICION HASTA QUE A=(HL) O BC=0	
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	CARGA	
LDI	DE ← DE+1 HL ← HL+1 BC ← BC-1		
LDDR	DE ← DE-1 HL ← HL-1 BC ← BC-1	REPETICION HASTA QUE BC=0	DECREMENTO DESPUES DE LA EJECUCION
LDIR	DE ← DE+1 HL ← HL+1 BC ← BC-1	REPETICION HASTA QUE BC=0	DECREMENTO DESPUES DE LA EJECUCION
IND	(HL) ← (C) B ← B-1 HL ← HL-1	BLOQUES, REPETICION (ENTRADA, SALIDA) BC CONTIENE LA DIRECCION DE LA VIA DE ACCESO (PORT) DE ENTRADA HL CONTIENE LA DIRECCION DE LOS DATOS	
INI	B ← B-1 HL ← HL+1		
INDR	B ← B-1 HL ← HL-1	REPETICION HASTA QUE B=0	DECREMENTO ANTES DE LA EJECUCION
INIR	B ← B-1 HL ← HL+1	REPETICION HASTA QUE B=0	DECREMENTO ANTES DE LA EJECUCION
OUTD	(C) ← (HL) B ← B-1 HL ← HL-1	BC CONTIENE LA DIRECCION DE LA VIA DE ACCESO (PORT) DE SALIDA HL CONTIENE LA DIRECCION DE LOS DATOS	
OUTI	B ← B-1 HL ← HL+1		
OTDR	B ← B-1 HL ← HL-1	REPETICION HASTA QUE B=0	DECREMENTO ANTES DE LA EJECUCION
OTIR	B ← B-1 HL ← HL+1	REPETICION HASTA QUE B=0	DECREMENTO ANTES DE LA EJECUCION

Figura 3.1d

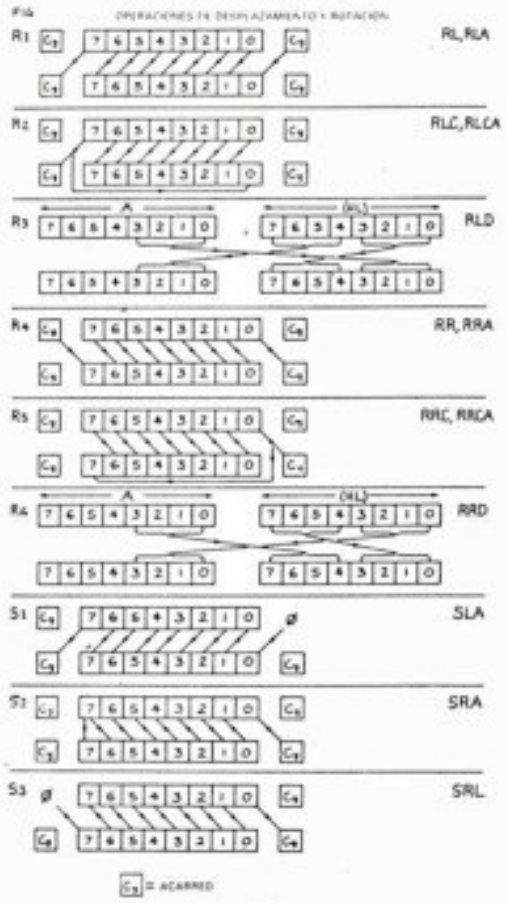


Figura 3.2

INSTRUCCIONES QUE ALTERNAN LOS SEÑALIZADORES (FLAGS)		SEÑALIZADORES EN EL REGISTRO F: S	Z		N		P/V	N	C_y	COMENTARIOS
ARITMETICA										
8 BIT	ADD A, r, ADC A, r	*	*		*		V	Ø	*	
	SUB r , SBC r	*	*		*		V	1	*	
	CP r	*	*		*		V	1	*	SEÑALIZADORES A = 1 PARA A =
	NEG	*	*		*		V	1	*	SEÑALIZADORES A = 1 PARA A = A
16 BIT	ADD							Ø	*	
	ADC	*	*				V	Ø	*	
	SBC	*	*				V	1	*	
LOGICA	AND r	*	*		1		P	Ø	Ø	
	OR r , XOR r	*	*		Ø		P	Ø	Ø	
	CPL				1			1		A = Ā
ROTACION	RLA , RRA				Ø			Ø	*	ROTACION DE A Y
	RLCA , RRCA				Ø			Ø	*	ROTACION DE A Y C_y
	RL r , RR r	*	*		Ø		P	Ø	*	ROTACION DE R Y
	RLC r , RRC r	*	*		Ø		P	Ø	*	ROTACION DE R Y C_y
DESPLAZAMIENTO	SLA r , SRA r	*	*		Ø		P	Ø	*	
	SRL r	*	*		Ø		P	Ø	*	
COMPROBACION DE BIT	BIT b, r		*		1			Ø		EL BIT B DE [illegible]
TRANSFERENCIA A ENTRADAS SALIDAS	IN r, (C)	*	*		Ø		P	Ø		
	INI , IND	$	*					1		ENTRADA SALIDA DE BLOQUES
	OUTI , OUTD	$	*					1		
	INIR , INDR	$	1					1		$ = Ø IF B ≠ Ø
	OTIR , OTDR	$	1					1		$ = 1 IF B = Ø
BLOQUES TRASLADO	LDI , LDD				Ø		#	Ø		# PUESTO A [illegible]
	LDIR , LDDR				Ø		Ø	Ø		
BUSQUEDA	CPI , CPD		$		*		#	1		$ PUESTO A [illegible]
	CPIR , CPDR		$		*		#	1		# PUESTO A [illegible]
OTRAS	CCF							Ø	$	ACARREO ACARREO
	DAA	*	*		*		P		*	AJUSTAR EL RESULTADO PARA CONTINUAR CON ARITMETICA ACD
	DEC r	*	*		*		V	1		
	INC r	*	*		*		V	Ø		
	LD A, I	*	*		Ø		$	Ø		EL «FLIP FLOP» DE HABILITACION DE INTERRUPCIONES SE TRASLADA A [illegible]
	LD A, R	*	*		Ø		$	Ø		
	RLD , RRD	*	*		Ø		P	Ø		
	SCF				Ø			Ø	1	ROTACION IZQUIERDA Y DERECHA [illegible]

SIMBOLOS UTILIZADOS EN LA TABLA

1. SIN SIMBOLO — NINGUNA ACCION
2. 0 — PUESTA A «0»
3. 1 — PUESTA A «1»
4. P — P/V CON CONTENIDO DEPENDIENTE DE LA PARIDAD DEL RESULTADO
5. V — P/V CON CONTENIDO DEPENDIENTE DEL DESBORDAMIENTO NEGATIVO DE BYTE
6. * — PUEDE ESTAR A «1» O A «0»
7. b — ES UN NUMERO DE BIT (DE 0 A 7)
8. r — ES UN REGISTRO SIMPLE O UN VALOR DE 1 BYTE
9. $ — } VER COMENTARIO EN LA SECCION «BLOQUES»
10. # — }

Figura 3.3

LA PILA («stack»)

La pila está formada por un grupo de elementos de memoria, de longitud variable, que puede almacenar datos mediante el sistema «LIFO» («last-in-first-out» = último en entrar, primero en salir). Es como una pila de cartas, la última que se coloca encima es la primera que debe quitarse. Pero para complicar un poco más las cosas, la pila se encuentra situada en la memoria en forma invertida, es decir, la parte alta de la pila (donde se introduce el último dato) está situada en una dirección más baja que la parte baja de la misma (donde se aloja el primer dato introducido). El puntero de pila («SP», «stack pointer») es un registro de 16 bits que indica siempre la dirección del último dato introducido en la pila.

Normalmente se utiliza la pila para almacenar direcciones de retorno de subrutinas, en forma de un par de bytes. Una instrucción CALL coloca el par en la pila (y decrementa el valor de SP en dos unidades) y otra instrucción RET recuperá el par de bytes (e incrementa el valor de SP en dos unidades). Sin embargo, existen también otras dos instrucciones que utilizan la pila: PUSH y POP. Cuando se utiliza PUSH con un registro de 16 bits, sus dos bytes son colocados en la parte alta de la pila y se decrementa el SP en dos. La instrucción contraria es POP, la cual coloca los valores de los dos últimos bytes de la pila en un registro de 16 bits, incrementando, al mismo tiempo, el valor de SP en dos unidades.

El uso repetido de PUSH reducirá eventualmente SP hasta que llegue a sobrescribir sus programas o datos y entonces pueden aparecer algunas consecuencias poco divertidas que suelen terminar en una puesta a cero del sistema.

Mientras que los POPs y los PUSHs se compensen entre si, el puntero SP no se descontrolará. Normalmente son suficientes unos 200 bytes para la pila aunque con subrutinas profundamente anidadas o con una programación más avanzada que la tratada en el presente libro, se necesitaría quizás algo más que esto. Recuerde que cuanto más alta sea la dirección inicial de su programa, menos espacio le quedará para la pila.

Si POP y PUSH se desequilibran durante la ejecución de una rutina, el resultado será que la subrutina no podrá retornar correctamente (un fallo muy común en los principiantes). Sin embargo, si al entrar en la subrutina se almacena el valor de SP en alguna otra dirección (¡que no pueda ser sobrescrita por la pila!), siempre podrá salirse correctamente del más profundo nivel de anidación de la

subrutina, mediante la restitución del valor de SP y la ejecución de la instrucción RET. Por ejemplo:

```
GRAFS    LD   (DIRECCION), SP
         ...
         ...
SALIDA   LD   HL (DIRECCION)
         LD   SP, HL
         RET
```

Todo lo que había en la pila sigue todavía allí, aunque no se utilice y sea después sobrescrito por las subsiguientes instrucciones PUSH.

Capitulo 4

LA REPRESENTACIÓN DE NUMEROS

«Cuando yo utilizo una palabra», dijo Humpty Dumpty en un tono despreciativo, «significa exactamente lo que yo he escogido que signifique, ni más ni menos.»
«La cuestión es», dijo Alicia, «cómo puede usted hacer que las palabras tengan significados tan diversos.»
«La cuestión es», dijo Humpty Dumpty, «cuál de ellos debe ser el correcto. Esto es todo.»
(Alicia a través del espejo.)

Dado el contenido de un byte cualquiera, poca cosa puede decirse de él aparte de su valor. Su significado depende del programador o del programa que dieron al byte su significado particular.

Ejemplo 1
Si el byte es una copia del registro F («Flags» = señalizadores), usted deberá fijar su atención en la *figura 3.3* y además posiblemente deberá retroceder en el programa para determinar qué operación y en qué dato alteró un bit determinado del byte.

Ejemplo 2
Puede ser parte de un número en el sistema de coma flotante del Spectrum (ver *figura 4.1*). Antes de asignar un significado al byte, debe determinar de cuál de los cinco bytes posibles se trata.

Ejemplo 3
Puede tratarse de un byte correspondiente a un número entero de 16 bits. De nuevo cabe preguntarse: ¿de cuál de los dos bytes posibles se trata?

Figura 4.1

Ejemplo 4

Puede ser un auténtico valor de un solo byte, como un carácter ASCII o bien un carácter propio del Spectrum. En este caso el significado puede deducirse de la *figura 4.2*. Nótese que si se utiliza un interfaz del tipo RS232/V24, necesitará seguramente insertar unos códigos de control de transmisión y disponer adecuadamente el bit más significativo de cada carácter ASCII de acuerdo con la paridad requerida por el periférico.

Ejemplo 5

Puede ser un código de instrucción o bien parte de él. Si empieza a descifrarlo desde un lugar incorrecto, el resultado será un galimatías.

Partiendo de un solo byte no hay manera de conocer lo que éste significa. Sin embargo, si se puede determinar la posición en la que se inicia el programa en código máquina, el resto del programa seguirá de una forma lógica y una vez en marcha su ordenación se establece automáticamente por el hardware.

LOS NUMEROS EN COMA FLOTANTE *(ver figura 4.1)*

Lea el manual del Spectrum (Capítulo 24). Lo que sigue son normas para trabajar con números en coma flotante.

El signo de la característica está en el byte con dirección más baja.

Cuando trabaje con números en coma flotante, debe siempre ajustar la magnitud del exponente de tal forma que el bit posterior al bit de signo de la característica sea el inverso del bit de signo, es

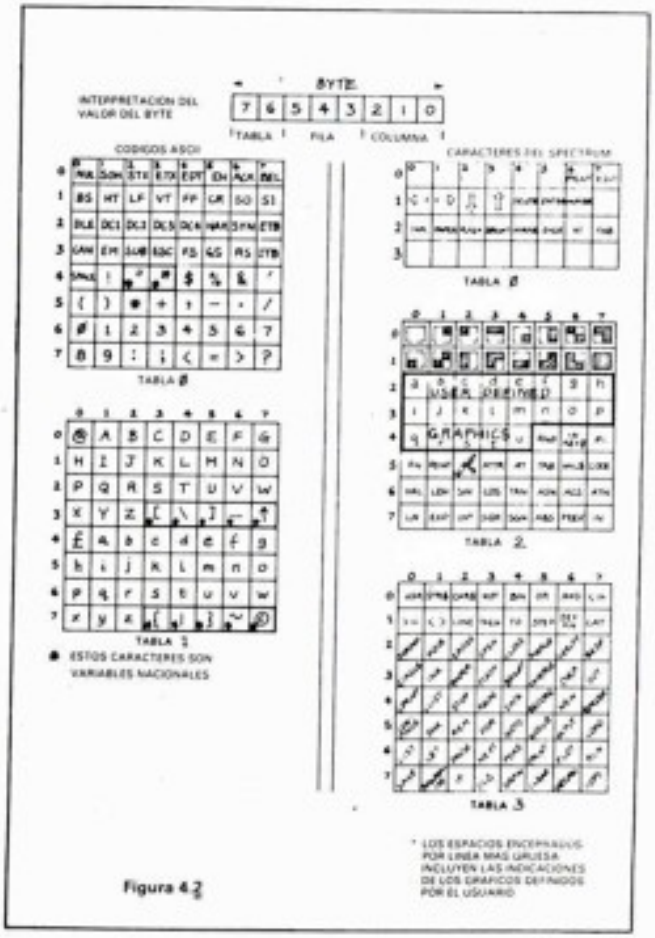


Figura 4.2

decir, que la característica puede empezar por 01 o bien por 10, pero nunca por 00 ni por 11.

Para sumar o restar números en coma flotante, primero deben igualarse los exponentes (desplace la característica del número menor hacia la derecha hasta que su exponente sea incrementado) y después sume o reste las características como sea preciso y corrija el exponente si es necesario. Este método de desplazamiento para igualar exponentes se llama normalización.

LA MULTIPLICACION Y LA DIVISION DE NUMEROS EN COMA FLOTANTE

1) No lo haga, a menos que se vea obligado a ello.
2) Si debe hacerlo:
   a) sume o reste los exponentes
   b) multiplique o divida las características; o bien:
   c) ¡deje al BASIC que lo haga por usted!

LAS ESTRUCTURAS DE DATOS

Las estructuras de datos pueden ser tan sencillas o complejas, largas o cortas como se desee, pueden enredarse o puede encontrarse un lugar para alojarlas. Hay una solución para cada tipo de problemas.

Supongamos que deba almacenarse una gran cantidad de datos alfanuméricos. Tenemos A-z, A-Z, 0-9, espacio y puntuación. Si introducimos un carácter SHIFT para distinguir entre mayúsculas y minúsculas y colocamos los números en el grupo contrario a la puntuación, podremos comprimir la totalidad del planteamiento en 40 códigos distintos. De esta forma, 40 * 40 * 40 = 64000 y 16 bits en 2 bytes tienen un valor máximo de 65535. Por el precio de algunos códigos podemos tener tres caracteres donde antes había sólo dos. Por tanto, logramos un incremento del 50 % en la capacidad de almacenamiento.

Tenemos ahora un caso parecido: existen en total 26 * 26 = 676 pares de letras posibles: aa, ab, ac, ... zx, zy, zz, los cuales no existen en su totalidad en ninguno de los idiomas más corrientes. Posiblemente, para una aplicación concreta, sean necesarios menos de 256 pares, con lo cual la introducción puede ser codificada a dos letras por byte, con el resultado de doblar la capacidad de almacenamiento.

Cuando se manejan grandes conjuntos o matrices de datos numéricos, de los cuales muchos elementos son nulos, debe procurarse encontrar la forma de tratar los datos no como matrices sino utilizando únicamente como los elementos no nulos. Esto será quizás algo más lento pero, al menos, resolveremos el problema.

LA ARITMETICA CON Y SIN SIGNO

La aritmética con signo utiliza el bit más significativo del valor para indicar el signo aritmético de los bits restantes. En la aritmética sin signo se sigue la pista de los valores de las variables. Generalmente basta con ignorar el bit de signo, como se hace en el direccionamiento (pero se tiene en cuenta el señalizador de acarreo («carry flag»)).

Capitulo 5

EL DIRECCIONAMIENTO

El direccionamiento es el método mediante el cual los datos o valores constantes almacenados en la memoria son transferidos a los registros del Z80 para trabajar con ellos, y es un concepto de gran importancia. El Z80 tiene muchos modos de direccionamiento, algunos más útiles que otros en ciertas aplicaciones. Debe usted tener siempre muy claro cuándo está utilizando variables de 8 bits y cuándo las usa de 16 bits. Las direcciones son siempre valores de 16 bits y se refieren o bien a un byte o bien al byte más bajo de los dos que forman el valor de 16 bits. Recuerde, no obstante, que en la zona de programa BASIC, el sistema del Spectrum tiene los bytes de los números de linea invertidos entre si.

Existen diferentes métodos para acceder a los datos. Algunos se detallan a continuación:

EL METODO DIRECTO

La posición del dato es conocida, y tiene un nombre o un valor numérico.

Por ejemplo:

LD HL,(23627)	colocará el contenido de 23627/8 en el registro doble HL.
LD A,(23627)	colocará el contenido del byte 23627 en el acumulador o registro A.

EL METODO DIRECTO CON UN DESPLAZAMIENTO FIJO

Por lo que se refiere a la forma de actuar, es idéntica al direccionamiento directo.

Por ejemplo:

```
LD   B,(PHRED+5)
```

PHRED es un valor determinado por el ensamblador durante el proceso de ensamblaje.

En la mayoría de los ensambladores, la dirección puede ser generada mediante cualquier combinación de etiquetas y valores numéricos unidos con signos «+» o «—». También puede darse un valor determinado a una etiqueta en vez de ser asignado este valor por el ensamblador.

EL METODO INDIRECTO

La dirección del dato requerido se encuentra en un lugar conocido. Por ejemplo:

```
LD   HL,(PHRED)
LD   B,(HL)
```

cargará HL con la dirección.
cargará B con el byte contenido en la dirección señalada por HL.

EL DIRECCIONAMIENTO INDEXADO

Este método utiliza dos registros de 16 bits, IX e IY. En este contexto, una página es una zona de no más de 256 bytes cuya dirección se carga como un valor de 16 bits en el registro IX o en el IY. Se supone que existen varias de estas páginas situadas ordenadamente conteniendo cada una de ellas datos para una utilización determinada (ver un ejemplo de ello en el Capítulo 10). Los datos son tratados mediante desplazamientos con relación al inicio de cada página.

Por ejemplo:

```
LD   A,(IX+5)
```

cargará A con el 6.° byte de la página que empieza en la dirección señalada por IX.

El método se vuelve más transparente si al desplazamiento fijo le damos un nombre que indique su contenido. Considérense, por ejemplo, los resultados de un examen. A cada alumno se le asigna una página organizada de la siguiente forma:

N.° de byte	*Contenido*	
0 }		
1 }	n.° de alumno	
2	nota	Matemáticas
3	»	Inglés
4	nota	Física
5	»	Historia
6	etc.	

Entonces podemos usar: LD A,(IX+FISICA), siempre y cuando le hayamos dicho al ensamblador que FISICA tiene el valor 4. Para cargar HL con el número de alumno debe ordenarse:

```
LD   L,(IX+0)
LD   H,(IX+1)
```

para cargar el byte «bajo»
para cargar el byte «alto»

Para pasar al siguiente alumno solamente será preciso sumar la cantidad adecuada al registro de página IX. En el manual del Spectrum se nos advierte que el registro IY no debe ser utilizado, pero esto no es estrictamente verdadero. Aunque su valor no debe alterarse *nunca*, está siempre fijado en 23610 y puede utilizarse para acceder a algunas variables del sistema. Por ejemplo, para poner a «1» el bit n.° 1 de los señalizadores («flags»), la instrucción sería:

```
SET  1,(IY+)
```

EL METODO INDIRECTO MULTIPLE

Cuando se tiene acceso solamente a la dirección del dato, hay que repetir el proceso utilizado para obtener una dirección indirecta. Teóricamente no hay límite en cuanto a los niveles a los que se puede llegar a actuar con este proceso, aunque no considero razonable más de tres niveles de descenso.

EL METODO ENCADENADO (ver *figura 5.1*)

Este método consiste generalmente en encadenar bloques de datos de tal forma que permita efectuar una localización rápida. El encadenamiento requiere que cada elemento de los datos lleve con-

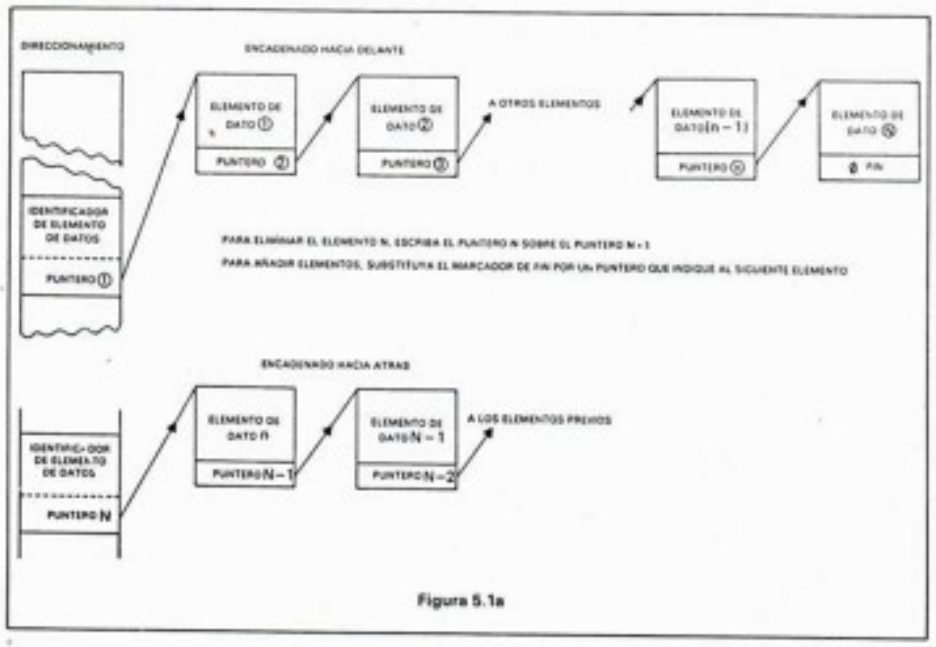


Figura 5.1a

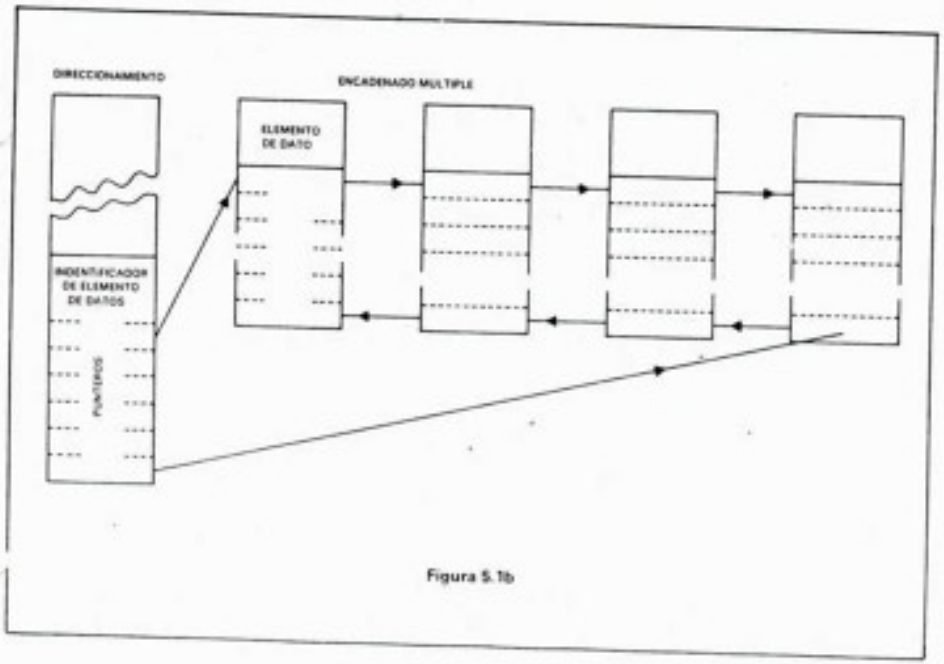


Figura 5.1b

sigo la dirección de uno o más elementos asociados. Estas direcciones se conocen también con el nombre de punteros. El encadenado puede ser hacia adelante, hacia atrás o bien ambos a la vez. La eliminación de un elemento de la cadena se realiza mediante un salto del puntero hacia otra dirección, de tal forma que no existen aquellos elementos que no están señalados por ningún puntero.

Normalmente es necesario utilizar una rutina que elimine la información inservible, ordene de nuevo los datos y borre físicamente los elementos que hayan sido eliminados de la cadena.

Nótese que varias cadenas independientes pueden enlazarse a través de los mismos datos (siempre que se destine espacio a sus punteros).

El programa BASIC del Spectrum es un ejemplo de encadenado hacia adelante. Cada línea lleva un puntero que señala el inicio de la línea siguiente. El encadenado hacia adelante es fácil. Los saltos hacia atrás (como GOTO... un número de línea anterior) son siempre búsquedas desde el principio.

LA COMPUTACION DE DIRECCIONES (E INSTRUCCIONES)

Cuando se trabaja con una gran variedad de direcciones, existe a veces la tentación de construir la dirección (o instrucción), introduciría directamente en el código y luego ejecutarla.

Esta técnica no es recomendable, aunque puede ser tolerada, especialmente cuando son importantes la velocidad y el tamaño del programa. Yo mismo la utilizo y todo lo que podría decirle es: «Tenga cuidado». Recuerde que no puede usar esta técnica si el programa va a ser cargado en un módulo PROM o ROM.

Notas:

1 Utilícelo solamente en subrutinas, nunca en la línea principal de un programa.
2 Al hacerlo en una subrutina, asegúrese de que conoce el efecto que producirá cada instrucción computada. Nunca compute una instrucción sin estar seguro de ello.
3 Asegúrese de que conoce la forma en que el ensamblador va a ensamblar las instrucciones que vayan a ser modificadas pues algunas de ellas pueden ser ensambladas de diferentes formas, por ej.: LD HL,(NN) puede ser codificado (en hexadecimal): 2A-n1-n2 o bien ED-6B-n1-n2. (Los ejemplos en código de este libro utilizan, a mi entender, un ensamblador que produce la más corta de dos formas equivalentes).
4 Si usted etiqueta la instrucción, entonces la etiqueta tiene la dirección del primer byte.
5 Recuerde, cuando documente o publique el código, prestar especial atención a lo que haya hecho. Otra persona con un ensamblador diferente puede producir un resultado distinto.

Capítulo 6

LOS PRINCIPIOS ELEMENTALES

INTRODUCCION

Este capítulo trata de dos aspectos esenciales de la programación en código máquina: los tiempos de ejecución de un programa y el traspaso de información entre subrutinas. Intento también mostrar la forma en que se desarrollan las soluciones. Lamento no conocer la fórmula para transferir años de experiencia a un principiante. A medida que usted vaya ganando experiencia, mire hacia atrás y observe sus primeros esfuerzos. Cuanto más deba retroceder para ello, más habrá aprendido.

EL BORRADO DE LA MEMORIA DE PANTALLA. UN ENSAYO EN TIEMPOS DE EJECUCION

Se trata de una rutina elemental para borrar los 6144 bytes de la memoria de pantalla (buffer), la cual empieza en la dirección 16384.

Mi primer intento fue éste:

```
        LD      BC,6144     1
        LD      HL,16384    2
CLRE    LD      A,0         3
        LD      (HL),A      4
        INC     HL          5
        DEC     BC          6
        LD      A,B         7
        OR      C           8
        JR      NZ,CLRE     9
```

y funciona pero no queda muy elegante.

Nótese, sin embargo:

a) Las líneas 7, 8 y 9 son un modo de comprobar que BC = 0 ya que las operaciones DEC o INC sobre registros dobles no afectan a los señalizadores.

b) Las instrucciones 3 y 4 pueden ser condensadas en una sola; olvidé que LD (HL),0 también es una instrucción válida.

El bucle 3/4 a 9 requiere 37 ciclos de reloj (ver *figura 3.1*) y se ejecuta 6144 veces, dando un total de 227.300 ciclos de reloj. ¿Podemos ganar rapidez?
2.ª versión:

```
        LD      HL,16384    1
        LD      C,24        2
LIN3    LD      B,0         3
LIN4    LD      (HL),0      4
        INC     HL          5
        DJNZ    LIN4        6
        DEC     C           7
        JR      NZ,LIN3     8
```

El bucle interior, el mayor consumidor de tiempo en rutinas de este tipo, requiere 6144 × 29 = 178.000 ciclos de reloj, lo cual representa un 78 % del tiempo exigido en el primer intento.

Sin embargo, aparecen problemas cuando se intenta generalizar la solución ya que en este caso se cumple que 6144 = 24 * 256, pero en otros, ¿estarían B y C dispuestos correctamente para las operaciones DEC y JR?

No hemos llegado todavía al final del camino. Lo que hemos hecho hasta ahora es cargar el mismo dato en distintas posiciones sucesivas. Supongamos ahora que primeramente borramos la posición 0, luego trasladamos la posición 0 a la posición 1, después la posición 1 a la posición 2, etc. Veamos qué ocurre utilizando, por ejemplo, la instrucción LDIR:

```
LD      HL,16384    1
LD      DE,16385    2
LD      BC,6143     3
LD      (HL),0      4
LDIR                5
```


y todo es efectuado 6143 veces por el LDIR a 21 ciclos de reloj cada vez. El tiempo total es, por tanto, 129.000 ciclos, o sea el 57 % del primer intento.

Si intentáramos cambiar los parámetros por variables y convertir esta subrutina en algo más universal, nos encontraríamos con todas las complejidades derivadas de sacar del juego a los parámetros, lo cual, hasta este momento, constituye un esfuerzo que no vale la pena realizar.

Con un pequeño cambio en la línea 4 tenemos la subrutina de borrado de pantalla CLRD («Clear Display») *(Listado 6.1)*, en la que entramos con A = 0.

LA PREPARACION DEL AREA DE ATRIBUTOS

Podemos aprovechar gran parte de la rutina anterior, sólo cambiando los valores de HL, DE y BC y dando un nuevo nombre a la rutina: SETA («Set Up Attributes»), la cual se utiliza con A = byte de atributo requerido.

Listado 6.1

```
1335 CLRD   PUSH AF
1340        PUSH BC
1345        PUSH HL
1350        LD   HL,16384
1355        LD   DE,16385
1360        LD   BC,6143
1365        LD   (HL),A
1370        LDIR
1375        POP  HL
1380        POP  BC
1385        POP  AF
1390        RET
```

Listado 6.2

```
0745 WAIT#  PUSH BC
0746    *   PUSH DE
0747        PUSH HL
0748        LD   BC,0
```

```
0749            LD    DE,0
0750            LD    HL,0
0751            PUSH  AF
0752            LDIR
0753            POP   AF
0754            POP   HL
0755            POP   DE
0756            POP   BC
0757            RET
0760 LWAIT      PUSH  BC
0761            LD    B,0
0762 LWAIU      CALL  WAIT#
0763            DJNZ  LWAIU
0764            POP   BC
0765            RET
```

No pretendo de ninguna manera asegurar que las rutinas de este libro tengan el mínimo de longitud o el mínimo tiempo de ejecución. Dos o tres personas en competición entre ellas deberían ser capaces de obtener importantes ahorros de tiempo y espacio en la mayoría de ellas.

LA RUTINA DE ESPERA (WAIT) Y EL TRASPASO DE DATOS ENTRE SUBRUTINAS

Cuando se utilizan programas en código máquina, ocurre frecuentemente que la presentación de datos en pantalla se produce de una forma más rápida de la que sería necesaria, ciertamente mucho más rápida de lo que puede ser asimilada. Necesitamos una rutina para retardar algo el proceso en estos casos.

Volviendo de nuevo a la rutina CLDR, la operación LDIR es relativamente lenta. Si la introdujéramos con HL = DE y BC = 0, consumiría un total de 65536 × 21 ciclos de reloj, o sea, aproximadamente 0,4 segundos a una velocidad del reloj de 3,5 MHz. Por lo tanto, podemos utilizar la rutina WAIT *(Listados 6.2 y 6.3)* procurando guardar los contenidos de los registros y recuperarlos después, para que la subrutina pueda ser llamada mediante la instrucción CALL WAIT desde cualquier punto sin temor a provocar una gran perturbación.

Si deseamos una pausa aún más larga, podemos colocar la llamada a WAIT dentro de otro bucle para obtener un tiempo de pausa de unos 60-70 segundos (rutina LWAIT).

Trabajando con la función WAIT, es interesante a veces que la rutina espere a que una tecla sea pulsada y, al mismo tiempo, podemos asignar misiones específicas a determinadas teclas (para prever su uso como entrada de datos, movimiento del cursor, juegos, etc.).

La respuesta al «¿cómo?» se encuentra en la posición 23560 del área de variables del Spectrum, que contiene precisamente el código de la última tecla pulsada por el usuario. Recuerde que el sistema de interrupción del Spectrum está funcionando durante todo el tiempo que sus rutinas están en marcha (usted está, realmente, trabajando a tiempo compartido con él), por lo tanto, podemos utilizar un bucle que lea el contenido de la dirección 23560 y espere a que aparezca en ella el código que deseamos.

Hay dos problemas a resolver:

a) ¿Cómo formamos la lista?
b) ¿Cómo le decimos a la rutina dónde se encuentra la lista?

Listado 6.3

```
0520 PAUSE   PUSH  AF
0525         PUSH  BC
0530         PUSH  DE
0535         PUSH  HL
0540 PAUS1   LD    A,(LASTK)
0545         CP    0
0550         JR    Z,PAUS1
0555         LD    (CHAR$),A
0560         LD    A,0
0565         LD    (LASTK),A
0570         POP   HL
0575         POP   BC
0580         POP   AF
0585         RET
0590 CHAR$   DEFB  0
0595 LASTK   EQU   23560
0600 IFKEY   POP   HL
0605         PUSH  HL
0610 IF1     LD    A,(LASTK)
0615         CP    0
0620         JR    Z,IF1
0625         LD    B,A
0630 NXBYT   LD    A,(HL)
 635         CP    0
```

```
0640            JR    Z,IFKEY
0645            CP    B
0650            JR    Z,MATCH
0655            LD    DE,4
0660            ADD   HL,DE
0665            JR    NXBYT
0670 MATCH      INC   HL
0675            POP   BC
0680            LD    B,A
0685            LD    A,0
0690            LD    (LASTK),A
0695            JP    (HL)
```

Comentario:

La lista debe contener dos cosas: un código de carácter y una dirección a dónde dirigirse cuando el carácter buscado aparezca. Algunos ensambladores no permiten colocar una dirección dentro de una lista y la dirección puede entonces encontrarse tan lejos que no pueda utilizarse un salto relativo para acceder a ella.

La lista debería tener esta forma:

Código de carácter
JP DIRECCION

¿Qué hay respecto a la longitud de la lista?

Podríamos determinar de antemano la longitud de la lista y colocarla en un registro dentro de la subrutina pero si luego deseáramos añadir o eliminar elementos de la lista, debería modificarse también aquel registro. Una forma mejor sería sacrificar un código de carácter y utilizarlo para indicar el fin de la lista. Yo uso el 0 ya que es poco utilizado y se puede comprobar muy fácilmente.

Ahora la lista aparecería de esta forma:

Código 1
JP DIRECCION 1
Código 2
JP DIRECCION 2
nop

La forma de la lista ha sido ya determinada. ¿Cómo le decimos ahora a la rutina dónde se encuentra la lista?

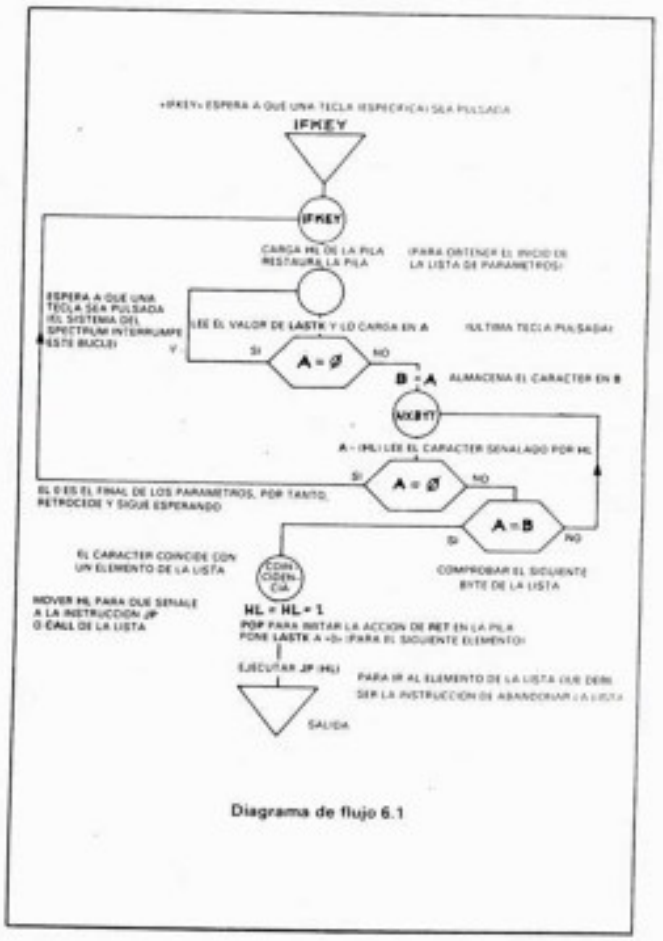


Diagrama de flujo 6.1

Nos encontramos ahora con dos formas de pensar distintas: una dice que las listas y constantes parecidas deben mantenerse apartadas y la otra sostiene que, siempre que sea posible, las constantes deben alojarse razonablemente cerca de las rutinas que tengan que utilizarlas.

Yo tiendo más hacia la segunda forma de pensar ya que, de esta forma, las rutinas quedan más auto-documentadas. Si llamamos pues a la rutina «IFKEY» (detección de tecla pulsada) – que tiene la tendencia de ser auto-explicativa –, su utilización podría ser como sigue:

```
CALL  IFKEY
DEFB  "A"
JP    AREAD
DEFB  "+"
JP    INCR
NOP
```

La instrucción DEFB coloca un código de carácter en la rutina. La llamada a IFKEY detiene el programa hasta que la A mayúscula o bien el símbolo « + » sean pulsados.

¿Cómo colocamos la lista en la rutina?

La parte más alta de la pila contiene la dirección de retorno de la subrutina. Por lo tanto, señala el código de «A» en el ejemplo anterior. Utilizamos POP para trasladarla desde la pila a un registro adecuado. Como usted ya habrá adivinado, la rutina IFKEY es llamada como una subrutina, pero *no* retorna al programa desde donde había sido llamada por medio de una instrucción RET (la cual requiere una operación POP suplementaria para compensar la acción PUSH inicial de la llamada CALL).

Sinopsis

CLRD	borra la pantalla.
IFKEY	espera hasta que una tecla de entre las que ha sido preseleccionada sea pulsada.
WAIT	produce una pausa de aproximadamente 1/4 segundo.
LWAIT	produce una pausa de aproximadamente un minuto.

Capítulo 7

LA PRESENTACION EN PANTALLA

El único medio real que posee el Spectrum para comunicarse con su usuario es a través de la pantalla del televisor y, por lo tanto, es muy importante saber controlar esta forma de comunicación. Primeramente voy a presentar la necesaria rutina de cálculo, seguida de un programa de presentación de caracteres completos.

Para presentar algo en pantalla, debemos antes conocer la forma de localizar un «pixel» determinado en la memoria de pantalla. Después continuaremos con la escritura de caracteres ASCII, cadenas de texto, números en base octal y presentación de los contenidos de los registros. Al mismo tiempo, se introducirá la idea de las «variables globales».

LA COLOCACION DE UN «PIXEL» EN LA MEMORIA DE PANTALLA

«La memoria de pantalla contiene una copia de la imagen del televisor...», como puede leerse en el Capítulo 24 del manual del Spectrum.

En todas estas rutinas, el origen de la pantalla se encuentra en la esquina superior izquierda.

La pantalla se divide en tres secciones, cada una de ellas con ocho líneas de texto (64 líneas de «pixels»). Hay 256 «pixels» en cada fila, contenidos en 32 bytes de datos. Un bit puesto a «1» significa que se trata de un elemento de tinta («ink»). Los 32 bytes siguientes a los de la fila 0 contienen los datos de la fila 8, los 32 siguientes, los de la fila 16, y así sucesivamente para el primer tercio de la pantalla (ver *figura 7.1a*).

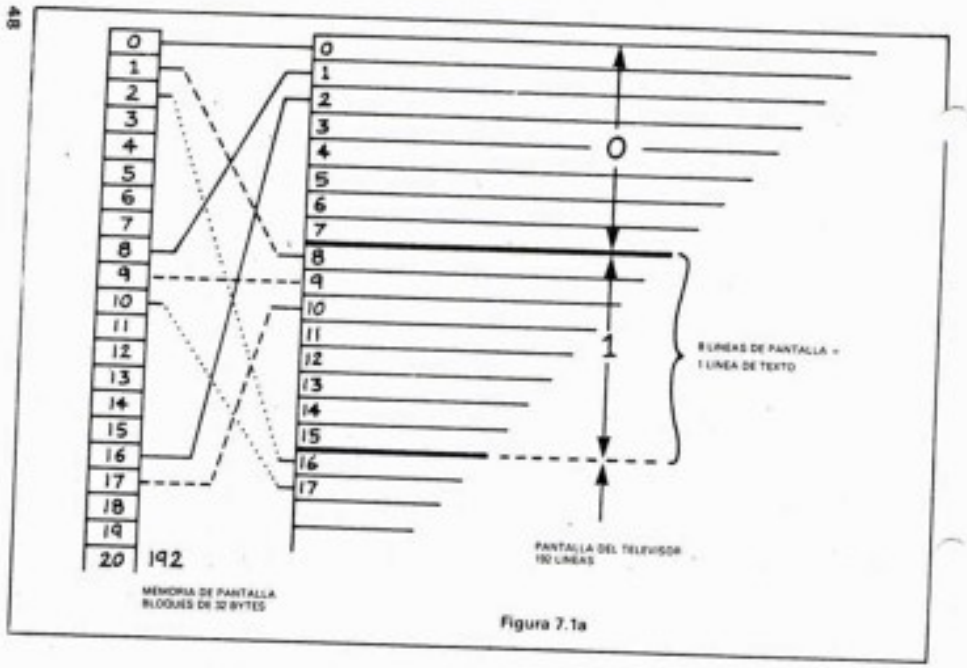


Figura 7.1a

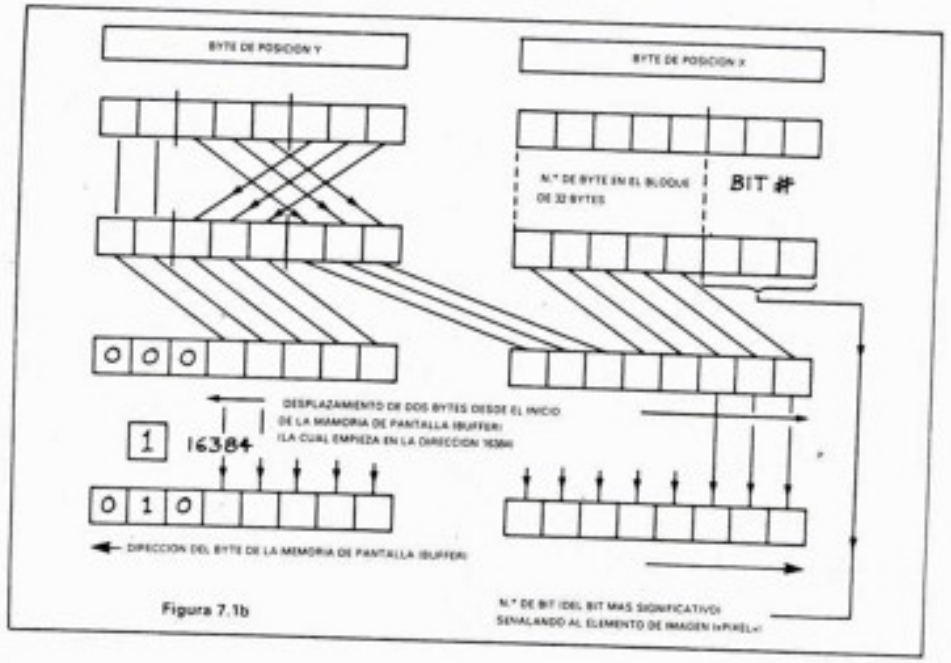


Figura 7.1b

De esto deducimos que la posición horizontal (o coordenada x) especifica un bit concreto dentro de los 256 posibles del bloque de 32 bytes.

El paso 1 consiste en colocar el valor x en un byte y utilizar los tres bits menos significativos para señalar un bit determinado dentro de un byte. Los cinco bits más significativos restantes indican concretamente a qué byte dentro del bloque de 32 se refiere.

El paso 2 consiste en determinar a cuál de los 192 bloques de 32 bytes debemos referirnos. Esto debe deducirse de la posición vertical (o coordenada y). En la *figura 7.1a* vemos que:

La fila 0 utiliza el bloque 0
La fila 1 utiliza el bloque 8
La fila 2 utiliza el bloque 16, etc.

Esto puede que no aclare mucho el asunto, escrito de esta forma, pero recuerde que estamos tratando con un ordenador y que si pensamos en binario o bien en octal podremos entendernos mucho mejor con él.

Listado 7.1

```
        0795 PLOT   PUSH BC
        0800        PUSH DE
a
        0805        LD   A,C
        0810        AND  7
b       0815        ADD  1
        0820        LD   E,A
        0825        SRL  C
c       0830        SRL  C
        0835        SRL  C
        0840        LD   A,B
        0845        AND  56
d       0850        SLA  A
        0855        SLA  A
        0860        OR   C
        0865        LD   C,A
        0870        LD   A,B
```



```
        0875        AND  7
        0880        LD   D,A
        0885        LD   A,B
e       0890        AND  192
        0895        SRL  A
        0900        SRL  A
        0905        SRL  A
        0910        ADD  D
f       0915        ADD  64
        0920        LD   B,A
        0925        PUSH BC
g       0930        POP  HL
        0935        LD   B,E
        0940        LD   A,128
h       0945        JR   PLA
        0950 PLB    SRL  A
        0955 PLA    DJNZ PLB
        0960        POP  DE
j       0965        POP  BC
        0970        RET
```

Repitamos ahora la relación de la *figura 7.1a* en octal:

La fila 00 utiliza el bloque 00
La fila 01 utiliza el bloque 10
La fila 02 utiliza el bloque 20
La fila 10 utiliza el bloque 01
La fila 11 utiliza el bloque 11
La fila 20 utiliza el bloque 02
La fila 21 utiliza el bloque 12

¡y se hizo la luz!

Para las 64 filas de cada sección, todo lo que debemos hacer es invertir entre sí los dos dígitos octales menos significativos del número de fila (que es la coordenada y) para tener indicado el bloque de 32 bytes de la sección. Los dos bits restantes del número de fila pueden ser entonces 00, 01 ó 10, para seleccionar cuál de las tres secciones queremos (11 no es un valor válido).

Ahora que sabemos lo que queremos hacer, podemos dibujar un diagrama de manipulación de bit *(figura 7.1b)*. A partir de aquí, el programa es relativamente sencillo pero observe que se opera solamente con registros. Cuando una rutina debe utilizarse muy frecuentemente, han de evitarse los accesos a la memoria ya que consumen la mitad más de tiempo que las operaciones directas con registros. Por otra parte, cuando deban escribirse caracteres en pantalla, esta rutina deberá llamarse ocho veces por cada carácter, o sea, 6144 veces para llenar la pantalla.

Vamos ahora a por las formalidades y la descripción del programa:

Rutina PLOT

Condiciones de entrada: posición x en el registro C
posición y en el registro B

Condiciones de salida: BC igual que a la entrada
DE igual que a la entrada
HL dirección del byte en la memoria de pantalla
A un bit puesto a «1» corresponde al bit en el byte direccionado por HL que se refiere al «pixel» definido por BC.

Notas:

1 El bit buscado en la memoria de pantalla está solamente indicado.
2 No hay diagrama de flujo debido a que el programa presenta una estructura muy directa (excepto para el desplazamiento del registro A).

Esto es un ejemplo de la documentación que había mencionado anteriormente (en el Capítulo 2). Vamos ahora con la descripción del programa.

SECCIÓN DESCRIPCION

a Guarda los registros en la pila.
b Enmascara los bits correspondientes al número de bit, añade una unidad y guarda el resultado en el registro E (ver la sección (h) más abajo para la razón de esta suma).
c Desplaza el contenido del registro C tres bits a la derecha (esto forma el índice para indicar la posición dentro del bloque de 32 bytes).

d 1.ª parte de la inversión octal (56 decimal = 70 octal); coloca el contenido de B en A, lo enmascara con 56 decimal, lo mueve dos lugares a la izquierda y coloca estos tres bits en los tres bits más significativos del registro C (junto con los cinco bits que indican el byte dentro del bloque).
e 2.ª parte de la inversión octal; extracción de los tres bits menos significativos del registro B y almacenamiento de los mismos en el registro D. 192 en decimal es 128 + 64, que son los valores de los dos bits más significativos de un byte. Se extraen estos bits del registro B (indican qué sección se necesita), se mueven tres lugares hacia la derecha y luego se suman a los tres bits del registro D.
f La memoria de pantalla empieza en la dirección 16384, que representa el bit 6 (64 en decimal) en el byte más significativo de una dirección de dos bytes. Se añade 64 al total del registro A y se guarda el resultado en B.

Nota: BC está ahora preparado con la dirección requerida (en el supuesto de que BC esté señalando un «pixel» válido para empezar). Queda todavía pendiente el problema de preparar el registro A.

g BC se transfiere a HL.
h B se carga con el contenido que E posee desde el paso (b). Hay una unidad más que en el cómputo de los tres bits menos significativos, ya que B se decrementa con la operación DJNZ antes del desplazamiento a la derecha.
El bit 7 es puesto a 1 en A, y el par de instrucciones PLA/PLB desplazan A hacia la derecha hasta que B alcanza el valor cero. La salida se produce con A conteniendo un bit a «1» en el lugar correcto.
j Recuperación de BC y DE; A y HL son preparados de la forma requerida.

LA SALIDA

Esta es, probablemente, la rutina más complicada de todo el libro. Se usa siempre que se deba presentar algo en pantalla y, a menos que usted entienda exactamente la forma en que trabaja, se encontrará con algunas dificultades más adelante.

El sistema del Spectrum permite al usuario de BASIC posicionar

el inicio de un texto usando AT, y saltar una línea cada vez que se ejecuta una nueva instrucción PRINT. En la próxima parte del programa, donde vamos a imprimir caracteres de varias formas en la pantalla, el vértice superior izquierdo de cada carácter de 8 × 8 «píxels» se localiza en la pantalla con dos variables de 1 byte: LINE (línea) y COLM (columna). Sus posiciones relativas no deben ser alteradas ya que se utilizan de forma conjunta para preparar el registro BC en el momento de llamar a PLOT para determinar qué bytes deben cargarse en la memoria de pantalla.

Para simplificar, COLM se incrementa en ocho unidades y cuando se desborda y se convierte en 0, se incrementa LINE en ocho. Cuando LINE señala fuera de la pantalla, se ajusta a 0 y la pantalla comienza de nuevo en el vértice superior izquierdo de la misma. La rutina NPAGE (nueva página) pone ambas variables a 0 y llama a la rutina de borrado de pantalla CLRD.

Listado 7.2(1)

```
0975 PRIN   PUSH AF
0980        PUSH BC
0985        PUSH DE
0990        PUSH HL
0995        SUB  32
1000        JP   M,PXL
1005        SUB  96
1010        JP   P,PXL
1015        ADD  96
1020        PUSH AF
1025        LD   BC,(COLM)
1030        CALL PLOT
1035        EX   DE,HL
1040        POP  HL
1045        LD   L,H
1050        LD   H,0
1055        ADD  HL,HL
1060        ADD  HL,HL
1065        ADD  HL,HL
1070        LD   BC,15616
1075        ADD  HL,BC
1080        LD   B,8
1085 RPRT   LD   A,(HL)
1090        INC  HL
1095        EX   DE,HL
1100        LD   (HL),A
1105        INC  H
1110        EX   DE,HL
1115        DJNZ RPRT
1120        LD   A,(COLM)
1125        ADD  8
1130        LD   (COLM),A
1135        JR   NZ,PXL
1140        LD   A,(LINE)
1145        ADD  8
1150        LD   (LINE),A
1155        ADD  64
1160        JR   NZ,PXL
1165        LD   A,0
1170        LD   (COLM),A
1175        LD   (LINE),A
1180 PXL    POP  HL
1185        POP  DE
1190        POP  BC
1195        POP  AF
1200        RET
1205 COLM   NOP
1210 LINE   NOP
```


Listado 7.2(2)

```
1395 NPAGE  CALL CLRD
1400        PUSH AF
1405        LD   A,0
1410        LD   (LINE),A
1415        LD   (COLM),A
1420        POP  AF
1425        RET
```

PRIN

Esta rutina hace aparecer en pantalla un solo carácter en la localización definida por LINE y COLM; también ajusta los valores de LINE y COLM para que señalen a la posición del próximo carácter.

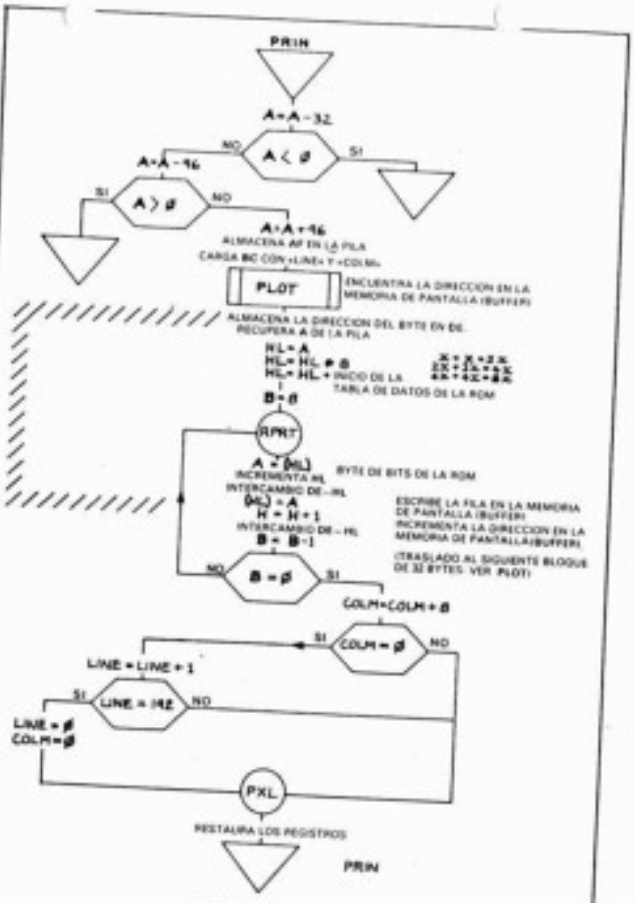


Diagrama de flujo 7.1

Esta rutina utiliza la tabla de caracteres contenida en la ROM del Spectrum, formada por matrices de bits, con ocho bytes por carácter para todos los códigos ASCII desde el 32 hasta el 127 inclusivos. Empieza en la dirección 15616 de la ROM y cada bloque de ocho bytes está dispuesto de la forma indicada por el manual del Spectrum, Capítulo 14.

Para imprimir un carácter en la pantalla, es necesario cargar los ocho bytes apropiados en la memoria de pantalla y ajustar los punteros LINE/COLM de forma que queden preparados para la impresión del siguiente carácter.

COMENTARIO

Esto funciona siempre que los punteros LINE/COLM no indiquen posiciones intermedias entre dos bytes o presenten segmentos de los límites de la pantalla. No veo ninguna razón para complicar el problema, pero de todas formas véase el Capítulo 8 para la forma de tratar el problema general.

La rutina PRIN se inicializa con el código ASCII del carácter en el registro A y se comprueba primeramente si es un carácter «PRINTable» o no lo es. Si no lo es, se omite y no es substituido por espacios. Se resta 32 del código ASCII para posicionar el puntero de los bytes de la ROM y se guarda A en la pila. Se utiliza luego PLOT para determinar la dirección del byte de la memoria de pantalla que ha de utilizarse para la primera fila de «píxels». LINE y COLM, guardados como bytes adyacentes, son recuperados conjuntamente con la instrucción LD BC,...

La dirección del byte que PLOT haya dado como resultado se almacena en DE y el código de carácter recuperado, multiplicado por 8 (8 bytes por carácter), y sumado a la dirección de inicio de la tabla de la ROM para indicar los bytes apropiados. Esta es la dirección que debe modificarse cuando usted desee utilizar su propia tabla de bytes de definición de caracteres.

Al registro B se le da el valor 8 para contar los 8 bytes que deben ser transferidos desde la ROM. Cada byte que se traslada a la memoria de pantalla produce además un incremento de 256 en el puntero para que éste señale al próximo bloque de 32 bytes de la pantalla donde debe colocarse el nuevo byte procedente de la ROM. Esto se lleva a cabo incrementando el registro H del registro doble HL en una unidad cuando contiene el dato apropiado. El bucle de transferencia en RPTR mantiene los punteros en HL y DE, intercambián-

dolos c..ndo es necesario. Comienza con HL señalando a la ROM y DE a la memoria de pantalla.

Una vez que el carácter ya ha sido escrito en la memoria de pantalla, COLM se incrementa en 8 (8 «pixels» forman una fila del carácter) para señalar al próximo carácter en la línea. Si la cuenta supera el máximo y se convierte en 0, se incrementa LINE en 8 (8 filas de «pixels» forman un carácter) y el resultado se compara con 192 para detectar si se ha llegado ya al final de la pantalla. Si se ha llegado a él, LINE y COLM son puestos ambos a cero. La rutina efectúa su salida después de recuperar los registros (excepto A) en PXL.

Esta rutina no trabajará correctamente si en algún momento LINE o COLM adoptan un valor que no sea múltiplo de 8. Un primer ejercicio para usted podría ser modificarla para asegurar que siempre sus contenidos sean múltiplos de 8.

PTEX

Imprimir textos («Print Text») es sólo una cuestión de alimentar PRIN con una secuencia de caracteres. Se consigue colocando el texto que debe ser impreso en los bytes inmediatamente siguientes a la llamada de la rutina y terminando la secuencia con un byte que contenga el valor cero. Esto funciona perfectamente bien con textos de una longitud fija, compuestos o asignados durante el proceso de ensamblaje pero será necesaria una versión modificada para trabajar con textos contenidos en algún otro lugar de dirección conocida. Le recomiendo encarecidamente, sin embargo, que utilice un byte con código cero para marcar el final de aquel texto ya que es un carácter que no puede ser impreso y es muy fácil su comprobación.

Listado 7.3(1)

```
1280 PTEX   POP   HL
1285 PXB    LD    A,(HL)
1290        CP    0
1295        JR    NZ,PXA
1300        JP    (HL)
1305 PXA    PUSH  HL
1310        LD    A,(HL)
1315        CALL  PRIN
1320        POP   HL
1325        INC   HL
1330        JR    PXB
```


Listado 7.3(2)

```
0700 SRHL$  SRL   L
0705        SRL   H
0710        RET   NC
0715        SET   7,L
0720        RET
0725 RHL3$  CALL  SRHL$
0730        CALL  SRHL$
0735        CALL  SRHL$
0740        RET
```

Vimos en IFKEY cómo se puede utilizar la dirección de retorno de la subrutina contenida en lo alto de la pila para acceder a los datos inmediatamente posteriores a la llamada a la subrutina. Utilizamos aquí el mismo sistema para acceder al primer carácter y a los siguientes de la cadena de texto que debe ser escrita. En PXB, con HL señalando un byte, es cargado en el registro A y comparado con cero. Si es el marcador de fin de texto, entonces HL señalará una instrucción NOP ya que la instrucción JP(HL) devuelve el control al programa principal. Si no es así, en PXA, A contendrá un carácter ASCII que ha de ser presentado por PRIN. Mientras PRIN está trabajando, el puntero de texto está guardado en la pila para que pueda recuperarse e incrementarse. Se efectúa luego un retorno a PXB para tomar el próximo carácter o bien el marcador de fin de texto.

Ahora vienen dos rutinas primitivas para efectuar el desplazamiento a la derecha de dos bytes (16 bits). Hay un método mejor, más elegante y más rápido para desplazar a la derecha.

SRHL

Las dos operaciones SRL desplazan cada registro un bit a la derecha. La segunda, en el byte más significativo, pondrá a «1» el señalizador de acarreo si se «pierde» algún bit del final o bien a «0» si no se pierde ninguno. La instrucción RET NC produce el retorno de la rutina cuando no debe hacerse ninguna corrección en el registro L; en otro caso, el bit perdido se substituye por el bit más significativo del registro L con la instrucción SET 7,L.

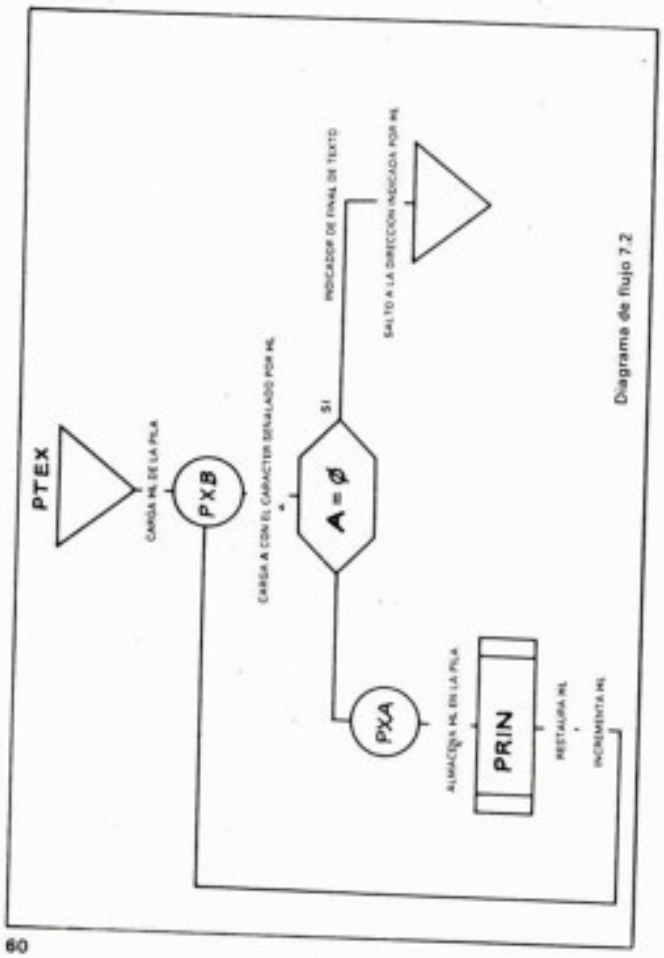


Diagrama de flujo 7.2

RHL3

Esta efectúa tres desplazamientos a la derecha juntos dividiendo, por lo tanto, el contenido del registro doble HL por 8, que es lo que se precisa cuando se imprimen números octales de la forma descrita en PRT8.

PRT8

Ahora ya podemos escribir textos pero, ¿qué hay de los números? Bien, existen una gran cantidad de rutinas complicadas para este fin en muchos otros sitios. Esta solamente va a escribir un número binario de 16 bits contenido en HL como un número octal de 6 dígitos. Esto se hace, en este caso, sin complicaciones ni signos. Es algo tan sencillo que podemos utilizarlo después como un método de eliminación de errores en los programas.

Hay dos trucos en ella:

1) Los caracteres ASCII de los números están dispuestos en forma secuencial a partir del código 48 (decimal) en adelante. Por tanto, el carácter octal requerido se obtiene sumando 48 a los tres bits binarios del valor en cuestión.

2) Utilizamos la rutina PTEX descrita anteriormente para presentar los datos y sobreescribir lo que se haya presentado anteriormente.

Al principio, se guardan todos los registros en la pila, con DE señalando al byte en el que el dígito menos significativo debe cargarse para ser impreso por PTEX. B toma el valor de 6 ya que sólo deben producirse 6 bytes. En PRU3, los tres bits menos significativos de HL se obtienen mediante el enmascarado y se calcula el código de carácter con la adición de valor 48. Esto se guarda en la posición indicada por DE, DE se decrementa y HL se desplaza tres bits a la derecha para dar paso al siguiente grupo octal si es que B no llega a cero. Si B llega a 0, significa entonces que los seis bytes ya han sido cargados en lo alto del espacio reservado «hgfedc» del listado. P8Z1 forma un par de espacios para completar la salida de los seis caracteres presentados por la llamada a PTEX, después de la cual todos los registros son restaurados a sus valores de entrada y la rutina efectúa su salida.

PRT8 puede ser, por tanto, insertado en cualquier lugar de un programa donde se necesite comprobar el contenido de HL.

Listado 7.4

```
1430 PRT8   PUSH AF
1435        PUSH BC
1440        PUSH DE
1445        PUSH HL
1450        LD   DE,PBZ1-1
1455        LD   B,6
1460 PRU3   LD   A,L
1465        AND  7
1470        ADD  48
1475        LD   (DE),A
1480        DEC  DE
1485        CALL RHL3#
1490        DJNZ PRU3
1495        CALL PTEX
1500        DEFM "cdefgh"
1505 PBZ1   DEFM "  "
1510        NOP
1515        POP  HL
1520        POP  DE
1525        POP  BC
1530        POP  AF
1535        RET
```

Listado 7.4(2)

```
5440 PRT8W  PUSH HL
5445        PUSH AF
5450        PUSH DE
5455        PUSH BC
5460        CALL PRT8
5465        CALL IFKEY
5470        DEFB "m"
5475        JP   PR8WX
5480        NOP
5485 PR8WX  POP  BC
5490        POP  DE
5495        POP  AF
5500        POP  HL
5505        RET
```

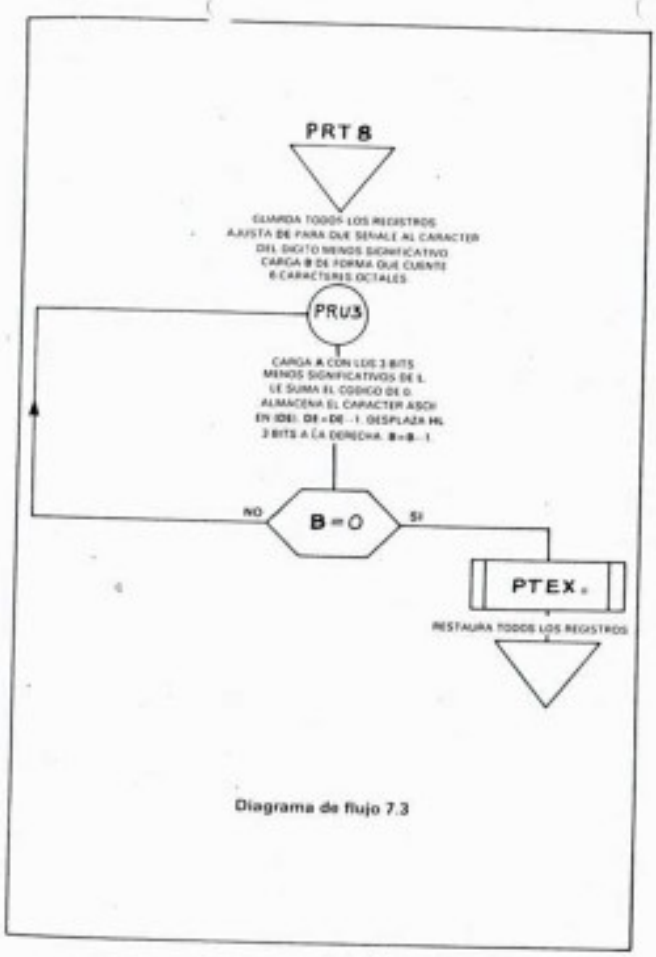



Diagrama de flujo 7.3

RPORT

En el proceso de eliminación de errores en los programas, aparece frecuentemente la necesidad de ver en la pantalla los contenidos de todos los registros. Esto es exactamente lo que hace la próxima rutina, mostrando además la dirección de retorno mediante PRT8. No hay diagrama de flujo por tener el programa una estructura muy directa.

Hay tres puntos a tener en cuenta:

1) El valor del puntero de pila, indicado por «$=» puede indicar si el programa se está desequilibrando debido a que los POPs y PUSHs no están reunidos.

2) La dirección de retorno de la llamada CALL RPORT, indicada por «£», permite distinguir entre varias salidas debidas a distintos puntos de llamada.

3) Los datos se cargan en HL para imprimir en pantalla la dirección de retorno. Se podría hacer de un modo mejor y más elegante computando la instrucción, como se demostrará más adelante en MOVER y VAR$1, por ejemplo.

Listado 7.5

```
1540 RPORT  LD    (SP#),SP
1545        PUSH  AF
1550        PUSH  BC
1555        PUSH  DE
1560        PUSH  HL
1565        LD    (HL#),HL
1570        LD    (DE#),DE
1575        LD    (BC#),BC
1580        PUSH  AF
1585        POP   HL
1590        LD    (AF#),HL
1595        CALL  PTEX
1600        DEFM  "AF="
1605        NOP
1610        LD    HL,(AF#)
1615        CALL  PRT8
1620        CALL  PTEX
1625        DEFM  "BC="
1630        NOP
1635        LD    HL,(BC#)
1640        CALL  PRT8
1645        CALL  PTEX
1650        DEFM  "HL="
1655        NOP
1660        LD    HL,(HL#)
1665        CALL  PRT8
1670        CALL  PTEX
1675        DEFM  "DE="
1680        NOP
1685        LD    HL,(DE#)
1690        CALL  PRT8
1695        CALL  PTEX
1700        DEFM  "#="
1705        NOP
1710        LD    HL,(SP#)
1715        CALL  PRT8
1720        CALL  PTEX
1725        DEFM  "£="
1730        NOP
1735        LD    HL,(SP#)
1740        LD    E,(HL)
1745        INC   HL
1750        LD    D,(HL)
1755        EX    DE,HL
1760        CALL  PRT8
1765        CALL  PAUSE
1770        POP   HL
1775        POP   DE
1780        POP   BC
1785        POP   AF
1790        RET
1795 AF#    DEFW  0
1800 HL#    DEFW  0
1805 DE#    DEFW  0
1810 BC#    DEFW  0
1815 SP#    DEFW  0
```

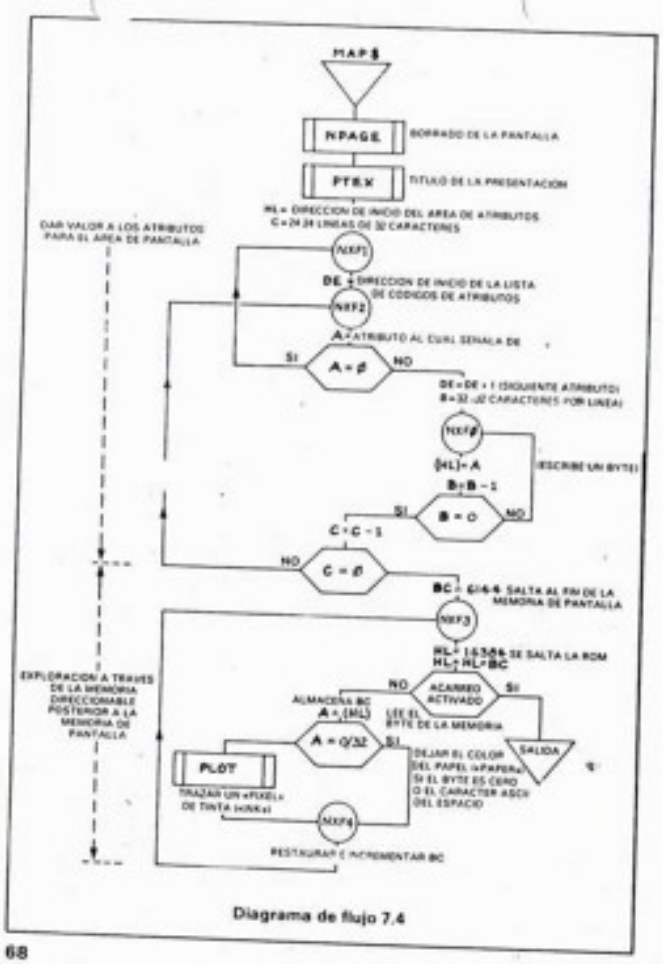


Diagrama de flujo 7.4

SINOPSIS

PLOT realiza la misma función que la instrucción PLOT del Spectrum. Es la base de toda presentación en pantalla. Forma también la parte básica de la rutina de animación del Capítulo 8 y el programa de dibujo del Capítulo 13.

PRIN imprime un carácter con un código ASCII en el registro A, en la siguiente posición disponible.

NPAGE borra la pantalla y prepara PRIN para que empiece en el principio de la primera línea.

PTEX utiliza PRIN para imprimir el texto que sigue a su llamada. (El texto debe terminar con un byte de valor cero.)

PRT8 utiliza PRIN para mostrar el contenido de HL como un número octal.

PRT8W utiliza PRT8 con una pausa de espera hasta que la tecla «ma «ea pulsada.

RPORT muestra los contenidos de los registros (excepto IX e IY).

MAP$ muestra la ocupación de memoria.

Listado 7.6

```
1820 MAP$   CALL NPAGE ;CLS
1825        CALL PTEX
1830        DEFM " ALMACENA MAPA DE MEMORIA
            DE 22528 A 65535 "
1835        NOP
1840        LD   HL,22528
1845        LD   C,24
1850 NXF1   LD   DE,LIST
1855 NXF2   LD   A,(DE)
1860        CP   0
1865        JR   Z,NXF1
1870        INC  DE
1875        LD   B,32
1880 NXF0   LD   (HL),A
1885        INC  HL
1890        DJNZ NXF0
1895        DEC  C
1900        JR   NZ,NXF2
1905        LD   BC,6144
1910 NXF3   LD   HL,16384
```

```
1915            ADD   HL,BC
1920            RET   C
1925            PUSH  BC
1930            LD    A,(HL)
1935            CP    0
1940            JR    Z,NXF4
1945            CP    32
1950            JR    Z,NXF4
1955            CALL  PLOT
1960            LD    B,(HL)
1965            OR    B
1970            LD    (HL),A
1975 NXF4       POP   BC
1980            INC   BC
1985            JR    NXF3
1990 LIST       DEFB  96
1995            DEFB  104
2000            DEFB  112
2005            DEFB  120
2010            NOP
```


Capítulo 8

LA ANIMACION

GCELL

El objeto de esta rutina es el de presentar una secuencia rápida de imágenes en un punto móvil de la pantalla. Estas imágenes están dibujadas en una caja o celda. Cuanto mayor sea la celda (hasta 2040 «pixels») más tardará la rutina. La comunicación de la rutina con el programa BASIC es compleja pero puede ser gestionada sin utilizar una técnica fundamentalmente nueva y más general (ver Capítulo 9).

LA COMUNICACION CON EL BASIC

El usuario debe cargar en los bytes 23675/6 (UDG = Gráfico Definido por el Usuario) la dirección del primer byte del bloque de datos definido más abajo que va a ser utilizado por la rutina. Puede haber varios bloques como éste y pueden conectarse entre si alterando el valor de la variable UDG.

BYTE	DESCRIPCION
0	posición horizontal de la celda X
1	posición vertical de la celda Y
2	señalizadores de control y número del cuadro siguiente
3	BCR : número de bits por fila de la celda (1 a 255)
4	BCC : número de bits por columna de la celda (1 a 255)
5	WPC : número de palabras por celda (1 a 255)
6	bytes de control de la secuencia de cuadros. Los cuatro bits menos significativos del byte n.° 2 señalan a uno de estos 15 bytes. Los cuatro bits me-

nos significativos de este byte definen qué celda debe dibujarse

20	byte para ser presentado
21	byte con valor cero
22	primer byte de datos de la celda 1. Hay WPC bytes en esta y en las otras celdas
22 + WPC	primer byte de datos de la celda 2
22 + 2 × WPC	primer byte de datos de la celda 3

y así sucesivamente para las demás celdas —hasta 15— que se precisen.

Descripción de los señalizadores de control y del número del siguiente cuadro —byte número 2:

N.º DE BIT	DESCRIPCION
7	Bit más significativo. Si está a «1», la rutina sale sin hacer nada.
6	Puesto a «1» por la misma rutina si alguna parte de la celda que se está dibujando sale fuera de la zona de pantalla permitida. Este bit debería ser monitorizado por el programa del usuario.
5 y 4	No utilizados.
3 a 0	Si son cero, la rutina retorna. Si no son cero, su contenido señala a un byte de control de la secuencia de cuadros que identifica la próxima celda que debe ser trazada. (Si se suma 5 al valor, el resultado dará de 6 a 20, que es el número de byte relativo al inicio del bloque, que contiene el identificador de la celda.) La rutina incrementa este puntero o lo restaura al valor 1 cuando se encuentra el final de la secuencia y reconoce al byte con valor cero que lo indica. Esto puede ser siempre sobreescrito utilizando el byte (UDG) + 2.

DATOS DE LA CELDA

Cada fila de «pixels», dentro de una celda de gráficos, empieza en el bit más significativo en una secuencia de bytes. Hay BCR/8 bytes en esta secuencia, y se ignoran los bits sobrantes. Hay BCC secuencias, una para cada fila de «pixels» de la celda. Cada bit puesto a «1» genera un «pixel» de tinta («ink») pero recuerde que el área de atributos debe ser puesta en orden en un ejercicio separado.

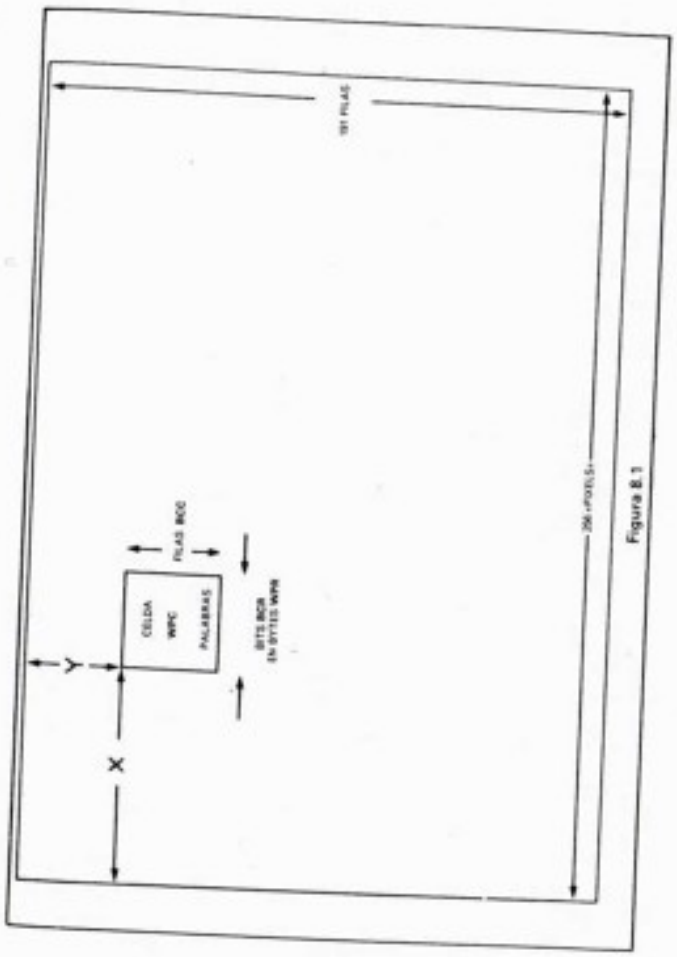


Figura 8.1

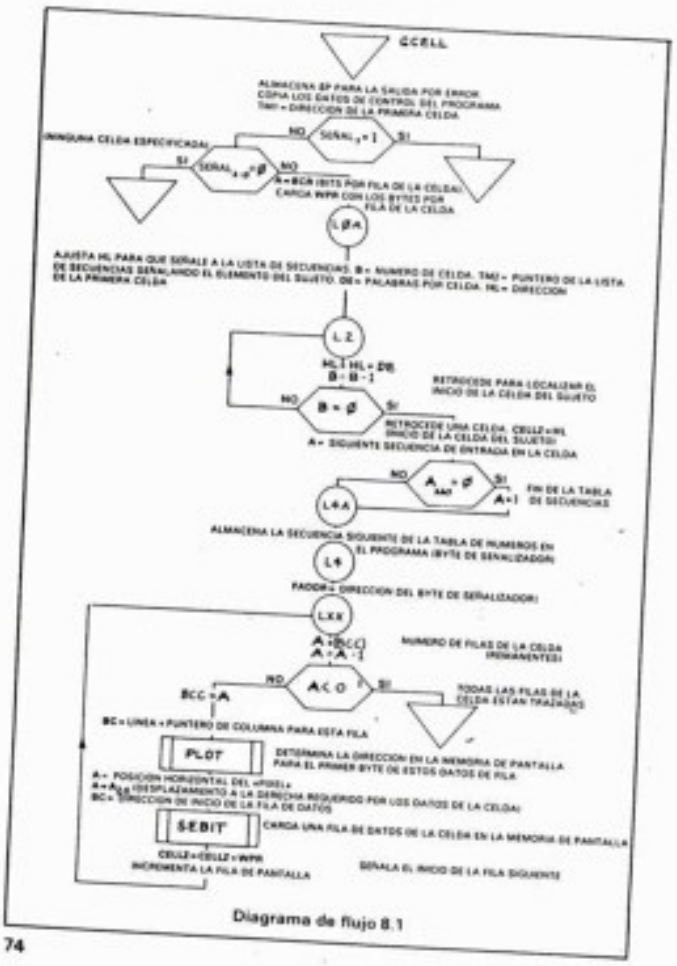


Diagrama de flujo 8.1

DESCRIPCION DE GCELL

El puntero de pila SP se guarda en PANIC. Por tanto, puede recuperarse en cualquier momento, permitiendo una salida digna a la rutina cuando, por ejemplo, ésta intente escribir más allá del área de memoria de pantalla permitida. Los primeros 21 bytes de los datos de control son copiados en la rutina y TM1 se iguala a la dirección de la primera celda gráfica. El byte 2, que se utiliza como byte señalizador, se comprueba y la rutina termina si el bit 7 está a «1» o bien si los bits 3 a 0 indican que ninguna celda está especificada. En otro caso, los últimos cuatro bits indican cuál de los bytes, en la tabla de secuencias, contiene el número de celda requerido. L0A a L2 y la operación DJNZ recogen este número de celda y colocan en CELLZ la dirección inicial de los datos de la celda.

En L4A — L4 se calcula el puntero de la próxima tabla de secuencias y se guarda en el byte n.° 2 de la tabla de comunicación con BASIC, donde queda preparado para ser utilizado en la próxima llamada a la rutina. En FADDR se coloca la dirección de este byte para que pueda utilizarla SEBIT si es necesario.

Listado 8.1

```
2015 GCELL  LD    (PANIC),SP
2020 UDG    EQU   23675
2025        LD    HL,(UDG)
2030        LD    DE,XY
2035        LD    BC,CELLZ-XY-1
2040        LDIR
2045        LD    (TM1),HL
2050        LD    A,(FLAG)
2055        BIT   7,A
2060        RET   NZ
2065        AND   15
2070        LD    (FLAG),A
2075        SUB   1
2080        RET   M
2085        LD    A,(BCR)
2090        SRL   A
2095        SRL   A
2100        SRL   A
2105        LD    (WPR),A
2110        LD    A,(BCR)
2115        AND   7
```

```
2120            JR    Z,L0A
2125            LD    A,(WPR)
2130            ADD   1
2135            LD    (WPR),A
2140 L0A        LD    HL,FSEQ-1
2145            LD    A,(FLAG)
2150            LD    B,0
2155            LD    C,A
2160            ADD   HL,BC
2165            LD    B,(HL)
2170            LD    (TM2),HL
2175            LD    A,(WPC)
2180            LD    E,A
2185            LD    D,0
2190            LD    HL,(TM1)
2195 L2         ADD   HL,DE
2200            DJNZ  L2
2205            OR    A
2210            SBC   HL,DE
2215            LD    (CELLZ),HL
2220            LD    HL,(TM2)
2225            INC   HL
2230            LD    A,(HL)
2235            AND   15
2240            JR    NZ,L4A
2245            LD    A,1
2250 L4A        LD    HL,(UDG)
2255            INC   HL
2260            INC   HL
2265            LD    (HL),A
2270 L4         LD    (FADDR),HL
2275 LXX        LD    A,(BCC)
2280            SUB   1
2285            RET   M
2290            LD    (BCC),A
2295            LD    BC,(XY)
2300            CALL  PLOT
2305            LD    A,(XY)
2310            AND   7
2315            LD    BC,(CELLZ)
2320            CALL  SEBIT
```



```
2325            LD    HL,(CELLZ)
2330            LD    BC,(WPR)
2335            ADD   HL,BC
2340            LD    (CELLZ),HL
2345            LD    A,(XY+1)
2350            ADD   1
2355            LD    (XY+1),A
2360            JR    LXX
2365 PANIC      DEFW  0
2370 XY         DEFW  0
2375 FLAG       DEFB  0
2380 BCR        DEFB  0
2385 BCC        DEFB  0
2390 WPC        DEFB  0
2395 FSEQ       DEFM  "cD.N.Laine 1983"
2400            NOP
2405 CELLZ      DEFW  0
2410 WPR        DEFW  0
2415 TM1        DEFW  0
2420 TM2        DEFW  0
2425 FADDR      DEFW  0
```

TRAZADO DE LA CELDA (DE LXX EN ADELANTE)

A la entrada, XY contiene la posición de la esquina superior izquierda de la celda en el formato requerido por PLOT. Las otras filas se dibujan incrementando la parte Y de XY. BCC se utiliza como un contador del número de filas que deben trazarse y la rutina sale cuando BCC, descende por debajo de cero.

PLOT determina la dirección del byte a cargar en la memoria de pantalla (buffer) y el número de bit para el inicio de la fila de «pixels» hacia la cual señala CELLZ. SEBIT traza la fila de «pixels». CELLZ es luego incrementada para señalar a la cabecera de la fila siguiente de «pixels» y el bucle se repite nuevamente.

SEBIT

Esta rutina traza o borra el trazado de «pixels» en la memoria de pantalla (buffer) según que los bits correspondientes estén a «1» o a «0» en la fila de la celda. Se mantienen dos punteros separados para actuar en ambos conjuntos de bytes.

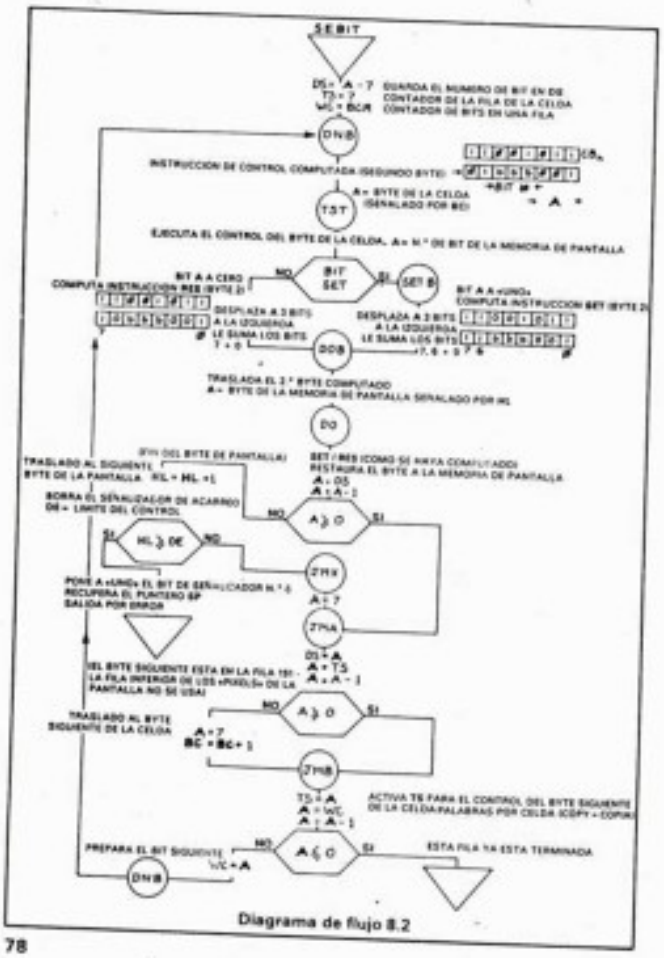


Diagrama de flujo 8.2

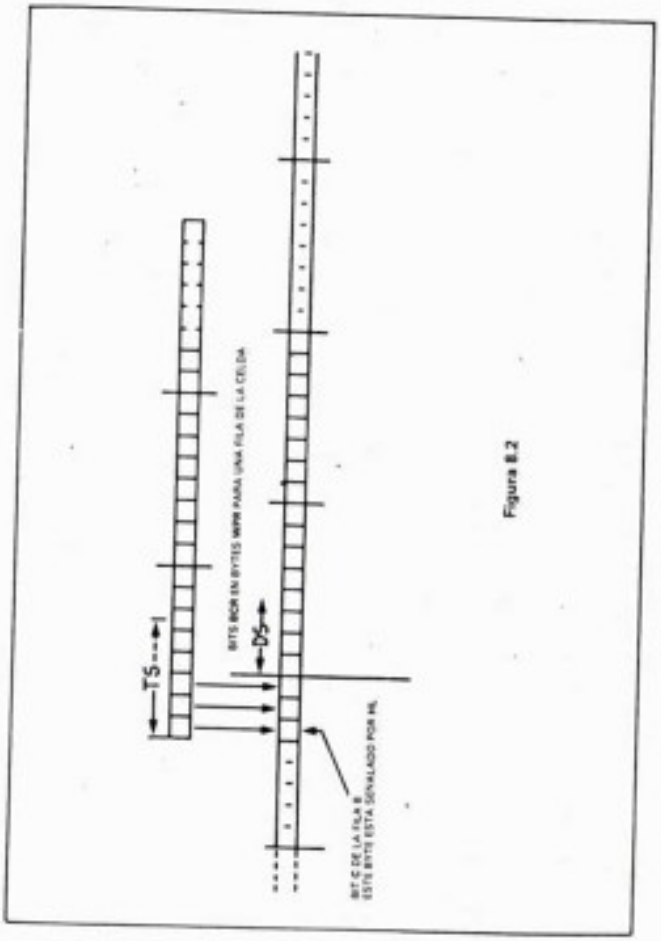


Figura 8.2

Al trar en DNB, HL señala a la memoria de pantalla (buffer) y DS señala al bit que debe ser puesto a «1» o bien a «0». CELLZ señala a los datos de la celda y TS es el número de bit del byte de la celda. WC es un contador de bits/«pixels» por fila de la celda (la rutina efectúa un bucle de BCR veces).

TST es una instrucción computada de control de bit para comprobar el bit TS en el byte de la celda según que el bit deba ponerse a «1» o a «0». Por tanto, las instrucciones SET o RES son computadas y ejecutadas en DO para poner a «1» o a «0» el bit requerido en la memoria de pantalla (buffer).

Al terminar con un «pixel», se incrementan los punteros. Si algún puntero de bit es negativo, se le asigna el valor 7 y se incrementa el correspondiente puntero de byte. Si el puntero de la memoria de pantalla, HL, señala la última fila de «pixels» entonces, para simplificar las cosas, se olvida esto. El byte que se utiliza como señalizador cambia su bit 6 al valor «1» y se efectúa la salida hacia la dirección de retorno contenida en PANIC.

Nota: En esta rutina se usan instrucciones computadas. ¿Cómo tratará su ensamblador éstas?:

64 + 7 genera la instrucción BIT ?, A
128 + 7 genera la instrucción RES ?, A
192 + 7 genera la instrucción SET ?, A

Sinopsis

GCELL, que usa PLOT, le permite hacer animación compleja. En el Capítulo 13 se mostrará la forma de preparar bloques de color. Más adelante aparecerá también una rutina para mover estos bloques.

Listado 8.2

```
2430 SEBIT  SUB   7
2435        NEG
2440        LD    (DS),A
2445        LD    A,7
2450        LD    (TS),A
2455        LD    A,(BCR)
2460        LD    (WC),A
2465 DNB    LD    A,(TS)
2470        SLA   A
2475        SLA   A
2480        SLA   A
2485        ADD   64+7
2490        LD    (TST+1),A
2495        LD    A,(BC)
2500 TST    BIT   1,A
2505        LD    A,(DS)
2510        JR    NZ,SETB
2515        SLA   A
2520        SLA   A
2525        SLA   A
2530        ADD   128+7
2535        JR    DOB
2540 SETB   LD    A,(DS)
2545        SLA   A
2550        SLA   A
2555        SLA   A
2560        ADD   192+7
2565 DOB    LD    (DO+1),A
2570        LD    A,(HL)
2575 DO     SET   0,A
2580        LD    (HL),A
2585        LD    A,(DS)
2590        DEC   A
2595        JP    P,JMA
2600        INC   HL
2605        OR    A
2610        LD    DE,22496
2615        SBC   HL,DE
2620        ADD   HL,DE
2625        JP    M,JMX
2630        LD    HL,(FADDR)
2635        LD    A,(HL)
2640        OR    64
2645        LD    (HL),A
2650        LD    HL,(PANIC)
2655        LD    SP,HL
2660        RET
2665 JMX    LD    A,7
```

```
2670 JMA    LD    (DS),A
2675        LD    A,(TS)
2680        DEC   A
2685        JP    P,JMB
2690        LD    A,7
2695        INC   BC
2700 JMB    LD    (TS),A
2705        LD    A,(WC)
2710        DEC   A
2715        RET   M
2720        RET   Z
2725        LD    (WC),A
2730        JR    DNB
2735 DS     DEFB  0
2740 TS     DEFB  0
2745 WC     DEFB  0
```


Capítulo 9

EL TRATAMIENTO DE ERRORES Y EL TRASPASO DE NOMBRES DE PARAMETROS

TRATAMIENTO DE LOS RETORNOS POR ERROR

En los comentarios acerca de la pila (Capitulo 3), mencionamos un método para salir de una rutina si aparece algún problema imprevisto e insoluble en un momento dado (véase también en el Capitulo 8 la utilización de PANIC). En casos como éstos es muy útil poder obtener alguna indicación sobre la naturaleza del problema.

El código máquina suele llamarse siempre mediante la instrucción RANDOMIZE USR ... Pero no existe una razón fundamental para hacer esto. En el manual del Spectrum, Capitulo 26, se utiliza la forma PRINT USR 32500 para visualizar el contenido del registro BC (cuyo valor ha sido asignado por el propio código máquina). Si tenemos, por ejemplo, una rutina que debe retornar normalmente con BC = 0, podemos llamarla con:

```
IF USR ... < > 0 THEN GOTO ... rutina de error
```

o mejor:

```
LET código de error = USR ...
IF código de error < > 0 THEN GOTO ...
```

ya que podemos darle cualquier significado especial al valor que no sea cero.

Todos estos IF ... < > 0 THEN GOTO ... son algo complejos y (peor aún) son en BASIC. Vea en el Capitulo 25 del manual del Spectrum las variables NEWPPC y NSPPC.

'PC nos dice exactamente lo que debemos hacer. Diseñamos la parte BASIC de nuestro programa de tal forma que alguna línea, la número 2, por ejemplo, sea la línea a la que haya que saltar en caso de una condición de error. Nuestra salida por error debe entonces contener:

```
LD HL,2
LD (23618),HL
LD A,1
LD (23620),A
```

que introduce el valor 2 en NEWPPC y el valor 1 en NSPPC. De esta forma se llega a la línea 2 del BASIC.

Volvemos ahora a llamar nuestra rutina con:

```
LET código de error = USR ...
```

y continuamos normalmente con la instrucción siguiente. Si aparece algún error, llegaremos a la línea 2 y la variable «código de error» contendrá el valor que tenía BC cuando la rutina efectuó su retorno. Aunque parezca extraño, la asignación de BC al código de error se produce sin importar la forma en que se efectúa el retorno.

Hay solamente un pequeño inconveniente en ello. BC puede utilizarse como un medio de traspaso de información entre el código máquina y el BASIC, y es una lástima sacrificar este medio para las rutinas de error. Después de todo, los buenos programas como los nuestros no suelen encontrar errores (!). ¿Podemos traspasar la información de error por algún otro medio?

De nuevo encontramos la respuesta en el manual del Spectrum, escondida en las profundidades de los Capítulos 24 y 25. Es un método algo más complejo pero puedo asegurar que vale la pena emplearlo.

1) 23627/8 — VARS contiene la dirección del inicio de la tabla de variables en el programa BASIC.

2) En el Capítulo 24 del manual se muestra la forma en que están organizadas estas variables y sus nombres.

Si la primera línea ejecutada en BASIC es:

```
0001 LET ERROR = 0 : GOTO ...
```

entonces el inicio del área de variables estará como la *figura 9.1* y

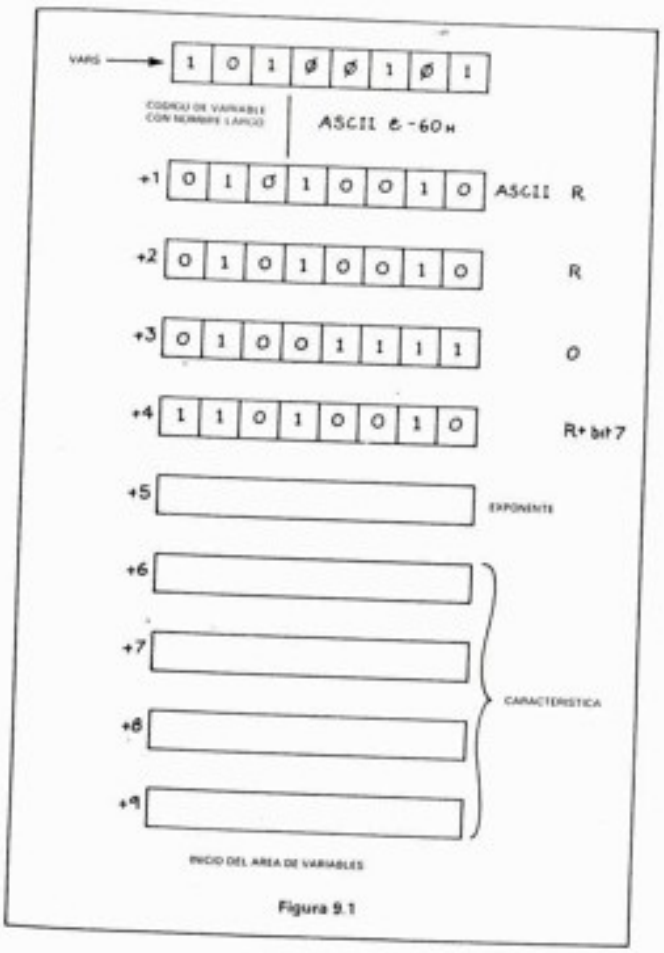


Figura 9.1

la rut... de error en el programa de código máquina puede localizar los bytes 5 a 9 e insertar los valores requeridos en la variable «error». La clave está en que la *primera* sentencia en el programa obligue a la *primera* variable a estar localizada en el principio del área de variables. En cualquier momento de la ejecución de un programa en BASIC, el orden de las variables dentro del área de variables dependerá de la forma en que hayan sido asignadas mediante sentencias LET y del orden en que éstas hayan sido ejecutadas.

Nuestro programa tendrá ahora la siguiente forma:

```
0001 LET error = 0 : GOTO 100
0002 PRINT «CONDICION DE ERROR = »; error
0003 ... rutinas de tratamiento de errores...
0004
...
0100 REM el programa propiamente dicho empieza aqui
0101
...
0150 LET q = USR ...
```

y *q* será algún valor útil generado por la rutina y traspasado al BASIC por medio del registro BC.

Nótese:

1) El registro BC sólo permite traspasar un valor desde el código máquina al BASIC.
2) «Error» puede, si se desea, ser un número en coma flotante de 5 bytes en los valores de los enteros del Spectrum.
3) Podemos ahora comunicar entre el código máquina y una variable BASIC y, por extensión, con el programa BASIC.
4) Podríamos incluso cambiar la utilización de «error» y utilizarlo como una variable de entrada al código máquina.

Podemos detenernos en este punto o continuar desarrollando un método para traspasar nombres de variables y valores (parámetros) entre el código máquina y el programa BASIC del Spectrum.

Hemos de ser capaces de hacer dos cosas:

1) Traspasar los nombres de variables a la rutina.
2) Dado un nombre de variable, encontrar su dirección en el área de variables del programa BASIC.

Pero vamos a formalizar primero cómo vamos a tratar las diciones de error.

A la entrada en el programa debemos asegurarnos de que l... ...mera variable BASIC tenga un nombre de cinco caracteres, el cual será reservado para traspasar códigos de error. Suponemos que nuestras rutinas de tratamiento de errores tendrán su inicio en la linea 2.

Las rutinas en código máquina empezarán guardando el valor del puntero de pila de forma que pueda utilizarse para el mecanismo de salida.

A la salida se retornará al BASIC colocando el valor que contenga el registro doble DE en aquel momento en la primera variable BASIC, y forzará el retorno a la linea 2.

La llegada del código máquina al mecanismo de salida se hará conteniendo DE un valor adecuado y con la instrucción JP, CALL o JR que se considere conveniente.

La rutina aparece en el *Listado 9.1.* ERROR es el valor archivado de SP (puntero de pila) a la llamada de la rutina.

Listado 9.1(1)

```
1215 ERREX  LD   HL,(ERROR)
1220        LD   SP,HL
1225        LD   HL,2
1230        LD   (23618),HL
1235        LD   A,1
1240        LD   (23620),A
1245        LD   HL,(23627)
1250        LD   BC,7
1255        ADD  HL,BC
1260        LD   (ERADR+2),HL
1265 ERADR  LD   (ERADR+2),DE
1270        RET
1275 ERROR  DEFW 0
```

Listado 9.1(2)

```
0005        ORG  60000
0010        LD   (ERROR),SP
0015        PUSH AF
```

```
0020        PUSH BC
0025        PUSH DE
0030        PUSH HL
0035        PUSH IX
0040        CALL DRAWL
0045        JP   TRAP#
0050        LD   (ERROR),SP
0055        PUSH AF
0060        PUSH BC
0065        PUSH DE
0070        PUSH HL
0075        PUSH IX
0080        CALL SATTR
0085        JP   TRAP#
0090        LD   (ERROR),SP
0095        PUSH AF
0100        PUSH BC
0105        PUSH DE
0110        PUSH HL
0115        PUSH IX
0120        CALL BLOCK
0125        JP   TRAP#
0130        LD   (ERROR),SP
0135        PUSH AF
0140        PUSH BC
0145        PUSH DE
0150        PUSH HL
0155        PUSH IX
0160        CALL SORTF
0165        JP   TRAP#
0170        LD   (ERROR),SP
0175        PUSH AF
0180        PUSH BC
0185        PUSH DE
0190        PUSH HL
0195        PUSH IX
0200        CALL GCELL
0205        JP   TRAP#
0210        LD   (ERROR),SP
0215        PUSH AF
0220        PUSH BC
```



```
0225        PUSH DE
0230        PUSH HL
0235        PUSH IX
0240        CALL MAP#
0245        JP   TRAP#
0250        LD   (ERROR),SP
0255        PUSH AF
0260        PUSH BC
0265        PUSH DE
0270        PUSH HL
0275        PUSH IX
0280        CALL IVERT
0285        JP   TRAP#
0290        LD   (ERROR),SP
0295        PUSH AF
0300        PUSH BC
0305        PUSH DE
0310        PUSH HL
0315        PUSH IX
0320        CALL MOVEC
0325        JP   TRAP#
0330        LD   (ERROR),SP
0335        PUSH AF
0340        PUSH BC
0345        PUSH DE
0350        PUSH HL
0355        PUSH IX
0360        CALL SVERT
0365        JP   TRAP#
0370        LD   (ERROR),SP
0375        PUSH AF
0380        PUSH BC
0385        PUSH DE
0390        PUSH HL
0395        PUSH IX
0400        CALL DRAWA
0405        JP   TRAP#
0410        LD   (ERROR),SP
0415        PUSH AF
0420        PUSH BC
0425        PUSH DE
```

```
0430            PUSH HL
0435            PUSH IX
0440            CALL DEMO1
0445            JP   TRAP#
0450            LD   (ERROR),SP
0455            PUSH AF
0460            PUSH BC
0465            PUSH DE
0470            PUSH HL
0475            PUSH IX
0480            CALL DEMO2
0485            JP   TRAP#
0490 .TRAP#     POP  IX
0495 TRAO#      POP  HL
0500 TRAR#      POP  DE
0505 TRAS#      POP  BC
0510 TRAT#      POP  AF
0515            RET
```

EL TRASPASO DE NOMBRES DE VARIABLES (PARAMETROS)

En el Capítulo 25 del manual del Spectrum aparece la forma de resolver este problema. NXTLIN (en las posiciones 23637/8) contiene la dirección del inicio de la próxima línea de programa BASIC, después de la que contiene la instrucción LET ... = USR ... Podríamos colocar una lista de parámetros en esta línea, escondida del sistema BASIC por una sentencia REM.

Una llamada a la rutina en código máquina con parámetros podría ser, por ejemplo:

```
0175 LET y = USR 12345
0176 REM a,b : REM a y b son parámetros de USR 12345.
```

En el Capítulo 24 del manual se explica cómo pueden obtenerse los nombres. NXTLIN señala al byte más significativo del número de línea, por tanto, (NXTLIN) + 4 será la dirección del primer carácter de texto de esta línea. Retrocedemos un lugar en la línea buscando el código REM (= 234), y luego empezamos a buscar el nombre de la variable del que indicaremos si termina con una coma, dos puntos o bien un carácter «ENTER». Los espacios son ignorados y los enteros se detectan porque comienzan con un dígito...

VAR$1, UNA RUTINA PARA LA BUSQUEDA DE VARIABLES EN EL AREA DE VARIABLES

Es fácil dejarse llevar por la euforia al diseñar un programa. Las especificaciones aumentan rápidamente sin concretarse y después es más difícil eliminar los errores, de tal forma que el programa que nació como una buena idea, se convierte luego en una pesadilla y es eventualmente abandonado con una mezcla de disgusto y desespero.

Vamos a dejar a un lado, por el momento, el traspaso de valores numéricos y nombres de variables multicarácter y nos limitaremos a traspasar un número reducido de variables de una sola letra (que pueden ser variables simples, cadenas o conjuntos). Siempre estaremos a tiempo de complicar las cosas más tarde.

El *diagrama de flujo 9.1* es una reproducción del diagrama original. El *diagrama 9.2* es la versión final que corresponde al *listado 9.2*. El recuadro VAR$1 es otra rutina que busca los nombres en el área de variables *(diagrama de flujo 9.3)*. Retorna con A = 0 final de datos o bien con HL conteniendo la dirección del inicio del nombre de la variable y A = los tres primeros bits del código de este nombre. Véanse los detalles más adelante.

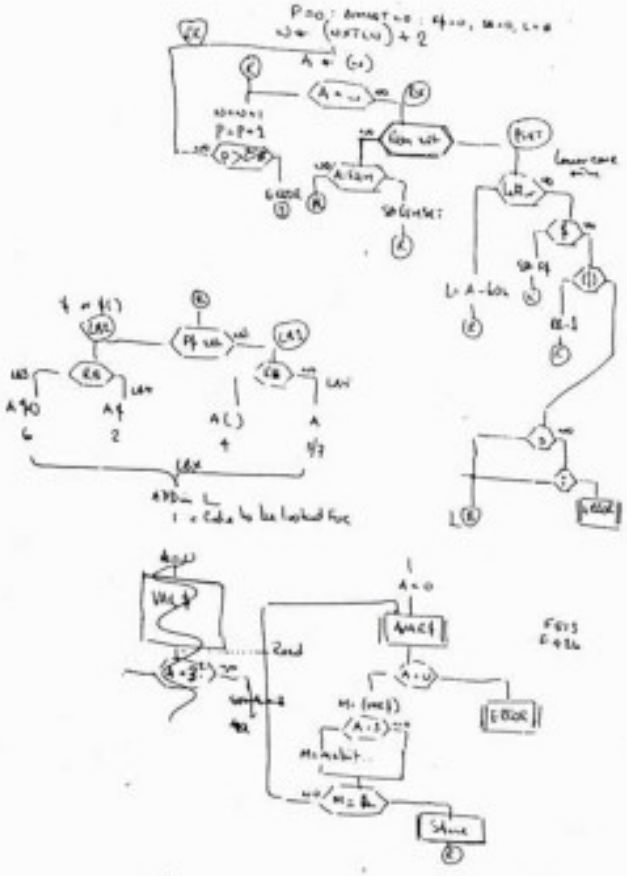
La rutina no hace exactamente lo que se creía que iba a hacer. Las letras, paréntesis, y $ pueden estar en cualquier orden y el identificador de variable es, en realidad, la última letra. La sentencia REM debe estar terminada con dos puntos o con un carácter «ENTER». Esto se deja para el lector, como un ejercicio sencillo para remediar estos defectos si lo desea.

Documentación de la rutina

Condiciones de entrada: ninguna

Condiciones de salida: destruidos los contenidos de todos los registros.
PARM0 — PARM6 son las direcciones de los primeros caracteres de hasta siete nombres de variables en el área de variables BASIC.
0 indica que no existe ningún parámetro.

Nota: La rutina que llama debe verificar que el tipo de variable recibida sea correcto y recoger o cargar los bytes apropiados.



Boceto del diagrama de flujo 9.1

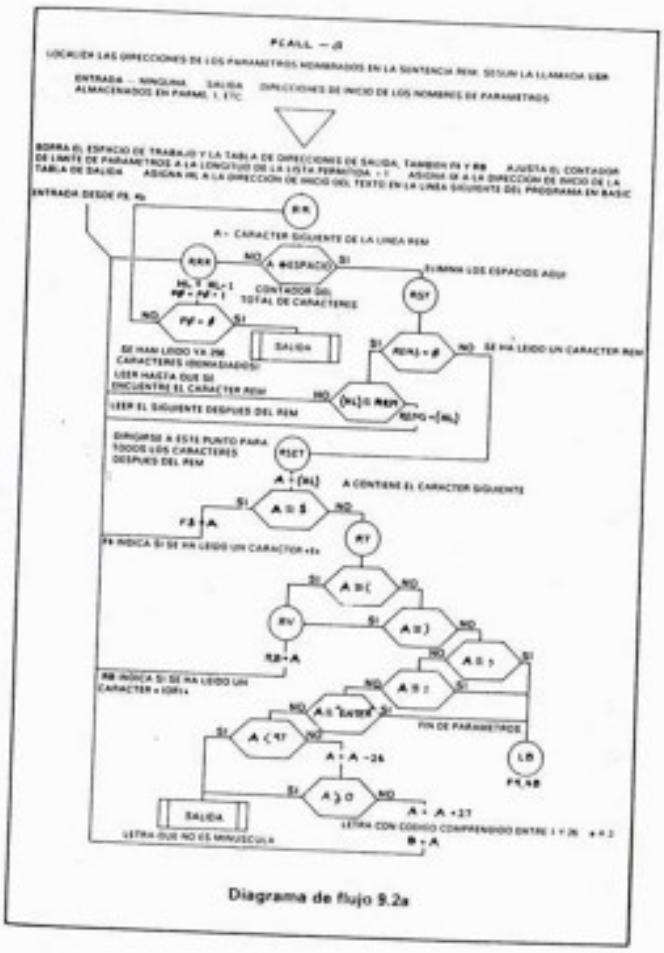


Diagrama de flujo 9.2a

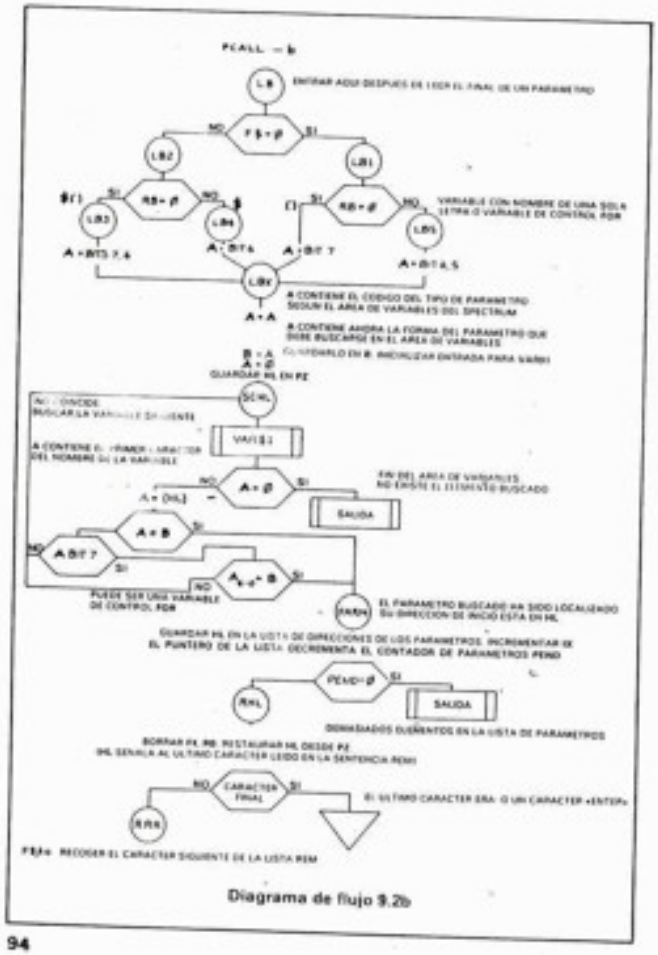


Diagrama de flujo 9.2b

PCALL

El número de parámetros que deben manejarse, PARM0, PARM1, ... se calcula durante el proceso de ensamblaje y PEND, su contador + 1 se ajusta a la entrada. El desplazamiento a la derecha permite alojar dos bytes en cada PARM. Para más parámetros, añádanse 2 bytes más por parámetro entre PARM6 y PEND; la operación LDIR borrará todo a la entrada. Recuerde, la lista PARM contiene las DIRECCIONES del inicio de cada nombre de variable.

La parte RR – RST – RRR de la rutina lee la siguiente linea de programa hasta que encuentra un carácter REM, y en este punto la variable REMS adopta un valor distinto de cero. Entonces RST salta a RSET, donde se comprobará el siguiente carácter de la sentencia REM que no sea un espacio. F$ adopta un valor no nulo si se encuentra un carácter $ indicador de cadena. RB se carga con un valor no nulo si se encuentra un carácter o bien indicadores de conjuntos («xarrays»). Una coma, dos puntos o un carácter «ENTER» fuerzan un salto a LB y entonces se comprueba si se trata de un carácter de letra minúscula. Si lo es, se guarda en B después de haberle restado 60 (hex). Cualquier carácter que falle en esta comprobación causa una salida por error que coloca el valor 2 en DE y llama a ERREX.

Cuando se alcanza LB, se ha leído un nombre de parámetro y el programa de LB a LBX examina F$ y RB para determinar el tipo y la forma –con la letra identificada– del inicio que debe buscarse en el área de variables.

A se pone a cero para inicializar VAR$1 en su primera llamada en SCHL. VAR$1 sale con A = 0 si no existen más variables y la rutina sale con error 2. En otro caso, HL señala el inicio del nombre de la variable que será comparada con el elemento buscado. Si no se encuentra todavía, el programa vuelve a SCHL con un valor no nulo en A, o, en caso contrario, PARM será el inicio del código que almacena la dirección de inicio de la variable en la lista de variables y antes se comprueba que haya sitio para ella. Los marcadores F$ y RB son puestos a cero y la rutina vuelve a RRR para leer el próximo carácter en la lista de parámetros o sale si se encuentra el final de la lista.

Listado 9.2

```
2750 PCALL   LD    HL,0
2755 NXTLN   EQU   23637
2760         LD    (PO),HL
```

```
2765
2770          LD    HL,P0
2775          LD    DE,P0+1
2780          LD    BC,PEND-P0
2785          LDIR
2790          LD    A,PEND+3-PARM0
2795          SRL   A
2800          LD    (PEND),A
2805          LD    IX,PARM0
2810          LD    BC,4
2815          LD    HL,(NXTLN)
2820 RR       ADD   HL,BC
2825          LD    A,(HL)
2830          CP    " "
2835 RRR      JR    NZ,RST
2840          INC   HL
2845          PUSH  HL
2850          LD    HL,P0
2855          INC   (HL)
2860          POP   HL
2865          JR    NZ,RR
2870          LD    DE,4
2875 RST      CALL  ERREX
2880          LD    A,(REMS)
2885          CP    0
2890          JR    NZ,RSET
2895          LD    A,(HL)
2900          CP    234
2905          JR    NZ,RRR
2910          LD    (REMS),A
2915 RSET     JR    RRR
2920          LD    A,(HL)
2925          CP    "$"
2930          JR    NZ,RT
2935          LD    (F$),A
2940 RT       JR    RRR
2945          CP    "("
2950 RV       JR    NZ,RU
2955          LD    (RB),A
2960 RU       JR    RRR
2965          CP    ")"
              JR    Z,RV
```



```
2970          CP    ","
2975          JR    Z,LB
2980          CP    ":"
2985          JR    Z,LB
2990          CP    13
2995          JR    Z,LB
3000          SUB   97
3005          JP    M,ERX2
3010          SUB   26
3015          JP    P,ERX2
3020          ADD   27
3025          LD    B,A
3030          JR    RRR
3035 LB       LD    A,(F$)
3040          CP    0
3045          JR    Z,LB1
3050 LB2      LD    A,(RB)
3055          CP    0
3060          JR    Z,LB4
3065 LB3      LD    A,128+64
3070          JR    LBX
3075 LB4      LD    A,64
3080          JR    LBX
3085 LB1      LD    A,(RB)
3090          CP    0
3095          JR    Z,LB5
3100          LD    A,128
3105          JR    LBX
3110 LB5      LD    A,64+32
3115 LBX      OR    B
3120          LD    B,A
3125          LD    A,0
3130          LD    (PZ),HL
3135 SCHL     CALL  VAR#1
3140          CP    0
3145          JP    Z,ERX2
3150          LD    A,(HL)
3155          CP    B
3160          JR    Z,PARM
3165          BIT   7,A
3170          JR    Z,SCHL
```

```
3175            AND   127
3180            CP    B
3185            JR    Z,PARM
3190            LD    A,1
3195            JR    SCHL
3200 PARM       LD    (IX+0),L
3205            LD    (IX+1),H
3210            INC   IX
3215            INC   IX
3220            LD    A,(PEND)
3225            DEC   A
3230            LD    (PEND),A
3235            JR    Z,ERX4
3240            LD    A,0
3245            LD    (F#),A
3250            LD    (RB),A
3255 RHL        LD    HL,(PZ)
3260            LD    A,(HL)
3265            CP    ":"
3270            RET   Z
3275            CP    13
3280            RET   Z
3285            JP    RRR
3290 P0         DEFB  0
3295 REMS       DEFB  0
3300 F#         DEFB  0
3305 RB         DEFB  0
3310 PZ         DEFW  0
3315 PARM0      DEFW  0
3320 PARM1      DEFW  0
3325 PARM2      DEFW  0
3330 PARM3      DEFW  0
3335 PARM4      DEFW  0
3340 PARM5      DEFW  0
3345 PARM6      DEFW  0
3350 PEND       DEFB  0
3355 ERX1       LD    DE,1
3360            CALL  ERREX
3365 ERX2       LD    DE,2
3370            CALL  ERREX
3375 ERX3       LD    DE,3
```



```
3380            CALL  ERREX
3385 ERX4       LD    DE,4
3390            CALL  ERREX
```

VAR$1

El área de variables consiste en una tabla ordenada de nombres y datos cuya dirección de inicio está dada por el contenido de las posiciones 23627/8. En el Manual del Spectrum se define el formato y la codificación de la tabla. VAR$1 utiliza esto para suministrar las direcciones de las variables.

Los tres primeros bits de cada variable definen su tipo y permiten acceder al inicio de la próxima variable. Estos bits son:

000 no utilizado
001 no utilizado
010 cadena
011 nombre de una variable de una sola letra
100 conjunto de números
101 nombre de varios caracteres
110 conjunto de caracteres
111 variable de control de un bucle FOR — NEXT

Estos códigos se utilizan para computar un salto relativo en JNV hacia otro salto relativo que trata de encontrar el final de una variable y el inicio de la siguiente. El proceso se inicia suponiendo que la variable previa no existente fuera del tipo 3, desplazando el inicio del área de variables en seis posiciones y saltando hacia J3.

Las etiquetas J0 a J7 identifican las secciones de código que tratan del tipo de variable correspondiente, como se indica en el *diagrama de flujo 9.3*. En el tipo 5, el final del nombre se indica poniendo a «1» el bit más significativo del último byte. FVEND busca esto último y la rutina procede como si se hubiera leído un nombre de variable de una sola letra.

Entre llamadas a VAR$1 la variable VAR$ mantiene la posición alcanzada durante la exploración. Durante la rutina, HL señala al área de variables.

En el caso de que la rutina empiece a generar errores después de haber funcionado correctamente, debe sospecharse que la variable VAR$ del Spectrum se ha alterado o bien que el área de variables ha sido sobreescrita.

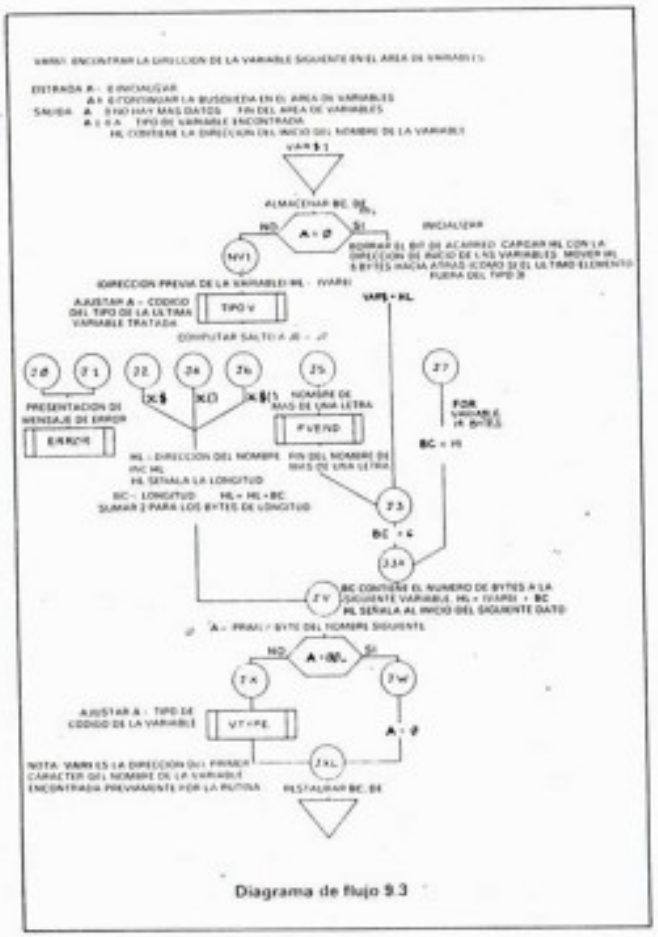


Diagrama de flujo 9.3

Nota sobre el salto computado en JNV:

La instrucción JR J0 es una instrucción de dos bytes
byte 0 = 24 decimal, 18 hex.
byte 1 = desplazamiento relativo.

Ya que la tabla entera consiste en saltos como estos, los desplazamientos requeridos serán 0, 2, 4, 6, etc. VTYPE sólo puede producir ocho valores (0 - 7). Si se desplaza un byte a la derecha, será 0, 2, 4, 6, 8, , 14 y la tabla cubrirá todas las posibilidades. La entrada 0 y la primera, ambas imposibles, saltan a la rutina de error.

Listado 9.3

```
3395 VARSS  EQU  23627
3400 VAR$1  PUSH BC
3405        PUSH DE
3410        CP   0
3415        JR   NZ,NV1
3420        OR   A
3425        LD   HL,(VARSS)
3430        LD   BC,6
3435        SBC  HL,BC
3440        LD   (VAR$),HL
3445        JR   J3
3450 NV1    CALL VTYPE
3455        LD   (JNV+1),A
3460        LD   DE,666
3465 JNV    JR   J0
3470        JR   J0
3475        JR   J0
3480        JR   J2
3485        JR   J3
3490        JR   J4
3495        JR   J5
3500        JR   J6
3505        JR   J7
3510 J0     CALL PREX
3515        DEFM "VAR$1 jo error"
3520        NOP
3525        CALL ERREX
3530 J2     INC  HL
```

```
3535          LD    (VP+2),HL
3540 VP       LD    BC,(VP+2)
3545          ADD   HL,BC
3550          INC   HL
3555          INC   HL
3560 JV       LD    (VAR#),HL
3565          LD    A,(HL)
3570          CP    0B0H
3575          JR    NZ,JX
3580 JW       LD    A,0
3585          JR    JXL
3590 JX       CALL  VTYPE
3595 JXL      POP   DE
3600          POP   BC
3605          RET
3610 J3       LD    BC,6
3615 J3X      ADD   HL,BC
3620          JR    JV
3625 J4       JR    J2
3630 J5       CALL  FVEND
3635          JR    J3
3640 J6       JR    J2
3645 J7       LD    BC,19
3650          JR    J3X
3655 VAR#     DEFW  0
3660 VTYPE    LD    A,(HL)
3665          AND   128+64+32
3670          RLC   A
3675          RLC   A
3680          RLC   A
3685          RLC   A
3690          RET
3695 FVEND    LD    HL,(VAR#)
3700          INC   HL
3705 FV1      BIT   7,(HL)
3710          JR    NZ,FV2
3715          INC   HL
3720          JR    FV1
3725 FV2      LD    (VAR#),HL
3730          RET
```


SINOPSIS

PCALL traspasa las direcciones de variables BASIC hacia las rutinas en código máquina.

Esto facilita el problema de traspaso de datos y la mayoría de rutinas del próximo capítulo utilizan ésta o la subrutina OPARS.

A la primera variable BASIC se le asigna el nombre ERROR y la linea 2 del programa BASIC se reserva para las rutinas de error.

Capítulo 10

RUTINA PARA ORDENAR NUMEROS EN COMA FLOTANTE

«Más allá de las montañas, la hierba es más verde.»
(Proverbio alemán)

UNA SOLUCION QUE PERMITE REALIZAR EN 145 SEGUNDOS LO QUE EXIGE CINCO HORAS EN BASIC

A aquellos lectores que hayan avanzado hasta estas alturas, déjenme presentarles una rutina útil y práctica: la ordenación de un conjunto o matriz de números en el formato de coma flotante del Spectrum. Se trata del método de ordenación por burbuja («bubble sort») prácticamente sin restricciones en cuanto al tamaño del conjunto. El tiempo de ejecución depende del cuadrado del número de entradas y es aproximadamente igual a $n^2/7000$ segundos para n entradas (unas 125 veces más rápida que su rutina equivalente en BASIC). Para 1000 entradas, tardará aproximadamente 145 segundos en ordenarlas, en vez de las cinco horas que tardaría en BASIC.

El método de burbuja no es el más rápido pero sí el más sencillo. Se comparan dos posiciones consecutivas en la tabla y la mayor, si no es la primera, se intercambia con la menor. Se va explorando repetidamente la tabla hasta que no sea preciso realizar ninguna inversión de elementos, lo cual significará que la rutina ya está en orden. En este punto la rutina efectuará su salida.

La primera pasada desde el principio hasta el final dejará siempre el valor mínimo al final. Si la siguiente pasada se realiza desde el final hasta el principio, el valor máximo se colocará al principio de la tabla. Con este sistema, la longitud de la parte no ordenada de la tabla se reduce continuamente a un elemento menos por cada

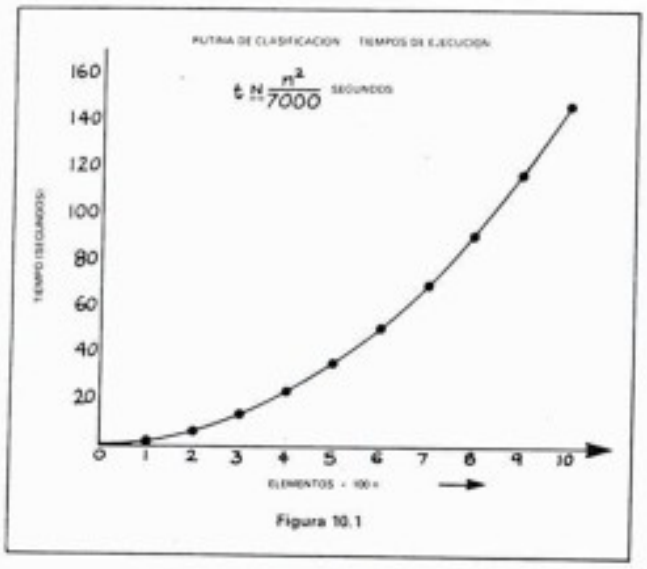


Figura 10.1

pasada, y el tiempo total puede reducirse casi en un 50 %. Le dejo esto para usted pues lo más difícil ya lo he hecho yo: el traspaso de los parámetros y la comparación de números en coma flotante.

SORTF

La rutina está dividida en dos partes. Primero la llamada a PCALL para obtener la lista de parámetros y la extracción de un parámetro localizado seguido de su comprobación. Si el parámetro no es del tipo 4 (conjunto de números), la rutina sale sin hacer nada. Entonces las variables HFE (inicio de la primera entrada) y HLE (inicio de la última entrada) son asignadas. Como no tenemos ninguna rutina de multiplicación, los saltos de la longitud del conjunto de los datos en el área de variables se realizan cargando el byte «número de dimensiones» en A y después sumándolo consigo mismo. ¡La restricción h

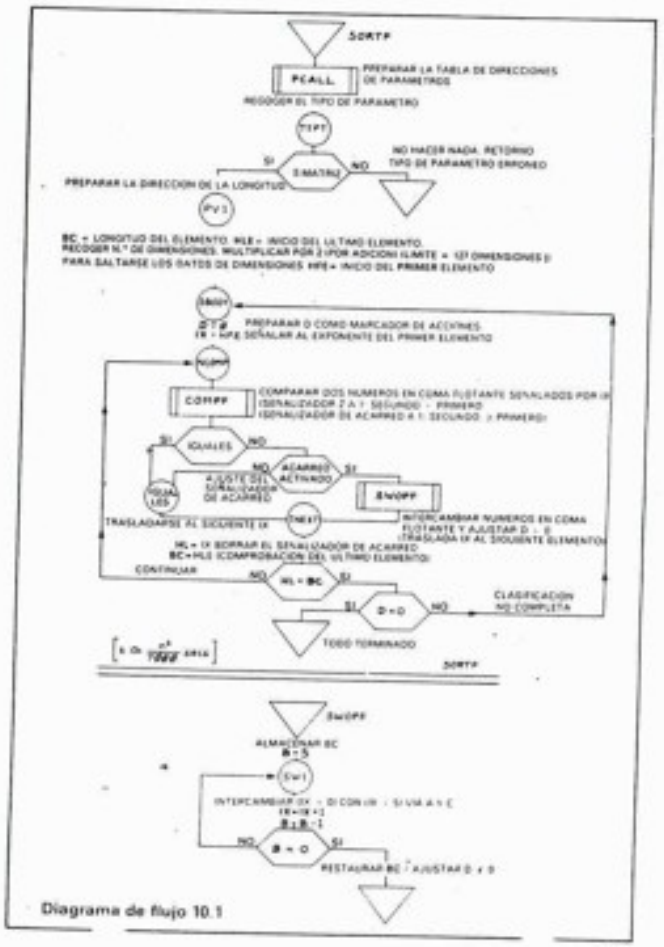


Diagrama de flujo 10.1

es que A no puede exceder de 127! Un conjunto de varias dimensiones se trata como si fuera un conjunto de una sola dimensión. El valor más alto se coloca en x(1,1, ...).

El cuerpo de la rutina SORTF tiene una estructura muy directa. El registro D se utiliza para indicar que se ha realizado una inversión y se carga en SWOPF. COMPF compara dos números consecutivos, y sale con el señalizador de cero (Z) a «1» si los números son iguales y con el señalizador de acarreo (c) a «1» si el segundo es mayor que el primero. Los dos números en coma flotante son consecutivos y el registro IX señala el byte de exponente con la dirección más baja.

Listado 10.1

```
3735 SORTF   CALL PCALL
3740         LD   HL,(PARM1)
3745         LD   (TYPT+1),HL
3750 TYPT    LD   A,(TYPT+1)
3755         AND  128+64+32
3760         CP   128
3765         RET  NZ
3770         INC  HL
3775         LD   (PV1+2),HL
3780 PV1     LD   BC,(PV1+2)
3785         ADD  HL,BC
3790         LD   BC,-3
3795         ADD  HL,BC
3800         LD   (HLE),HL
3805         LD   HL,(PARM1)
3810         LD   BC,3
3815         ADD  HL,BC
3820         LD   A,(HL)
3825         ADD  A
3830         LD   C,A
3835         LD   B,0
3840         ADD  HL,BC
3845         INC  HL
3850         LD   (HFE),HL
3855 SBODY   LD   D,0
3860         LD   IX,(HFE)
3865 NCOMP   CALL COMPF
3870         JR   Z,IGUALES
```



```
3875         JR   NC,IGUALES
3880         CALL SWOPF
3885         JR   TNEXT
3890 IGUALES LD   BC,5
3895         ADD  IX,BC
3900 TNEXT   PUSH IX
3905         POP  HL
3910         OR   A
3915         LD   BC,(HLE)
3920         SBC  HL,BC
3925         JR   NZ,NCOMP
3930         LD   A,D
3935         CP   0
3940         RET  Z
3945         JR   SBODY
3950 HFE     DEFW 0
3955 HLE     DEFW 0
3960 SWOPF   PUSH BC
3965         LD   B,5
3970 SW1     LD   A,(IX+0)
3975         LD   C,(IX+5)
3980         LD   (IX+0),C
3985         LD   (IX+5),A
3990         INC  IX
3995         DJNZ SW1
4000         POP  BC
4005         LD   D,1
4010         RET
```

COMPF

En la *Figura 10.2* se detalla el formato de un número en coma flotante. La rutina es bastante complicada y podría simplificarse mucho. IX señala al primer byte (de exponente) del primer número cuya mantisa está en IX + 1, 2, 3 y 4. El segundo número tiene su exponente en IX + 5 y su mantisa en IX + 6, 7, 8 y 9. Los signos de los números están en IX + 1 e IX + 6. Si son diferentes, el positivo es mayor que el negativo. Si son del mismo signo, se comparan sus exponentes. En la representación del Spectrum, los exponentes están desplazados en 128 y deben ser comparados y comprobar el señalizador de acarreo. El significado depende, sin embargo, del signo de

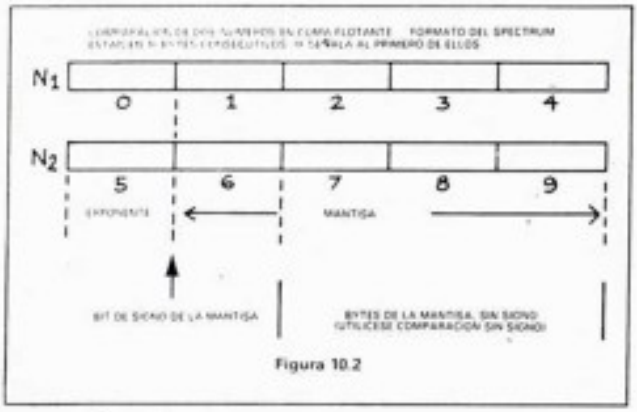


Figura 10.2

la mantisa. Con mantisas positivas, el exponente mayor pertenece al número mayor en coma flotante. Con mantisas negativas, el exponente mayor pertenece al número menor en coma flotante. El registro B adopta un valor no nulo para mantisas negativas.

Los números del mismo signo y exponentes iguales deben ser comparados byte a byte hasta que se detecte alguna diferencia entre ellos, si es que la hay. Los bytes de signo, al ser comparados, deben comprobarse con operaciones JP P o JP M, ya que el acarreo se produce solamente con un número negativo. Los bytes de mantisa restantes pueden comprobarse con el señalizador de acarreo puesto que no tienen signo. El significado de la decisión en BTL o BTG se decide por el signo de la mantisa, según el contenido del registro B. Si no se hiciera esta corrección, los números positivos se separarían de los negativos y se clasificarían los dos bloques por orden decreciente absoluto (sin tener en cuenta el signo).

Listado 10.2

```
4015 COMPF   LD   B,0
4020         BIT  7,(IX+1)
4025         JR   Z,CL1
4030 CL2     BIT  7,(IX+6)
4035         JR   Z,V1LV2
4040 CL3     LD   A,(IX+0)
4045         CP   (IX+5)
4050         JP   M,V1GV2
4055         JR   Z,CL4
4060         JR   V1LV2
4065 CL4     LD   B,255
4070         JR   XEQ
4075 CL1     BIT  7,(IX+6)
4080         JR   NZ,V1GV2
4085 CL5     LD   A,(IX+5)
4090         CP   (IX+0)
4095         JR   C,V1GV2
4100 CL6     JR   NZ,V1LV2
4105 XEQ     LD   A,(IX+1)
4110         CP   (IX+6)
4115         JR   Z,XEQM
4120         JP   P,BTG
4125         JR   BTL
4130 XEQM    LD   A,(IX+2)
4135         CP   (IX+7)
4140         JR   C,BTL
4145         JR   NZ,BTG
4150         LD   A,(IX+3)
4155         CP   (IX+8)
4160         JR   C,BTL
4165         JR   NZ,BTG
4170         LD   A,(IX+4)
4175         CP   (IX+9)
4180         JR   C,BTL
4185         JR   NZ,BTG
4190         RET
4195 V1GV2   LD   A,2
4200         CP   1
4205         RET
4210 V1LV2   LD   A,2
4215         CP   3
4220         RET
4225 BTL     BIT  1,B
4230         JR   NZ,V1GV2
```

```
4235          JR    V1LV2
4240 BTG      BIT   1,B
4245          JR    NZ,V1LV2
4250          JR    V1GV2
```

UN EJEMPLO PRACTICO DE SORTF

Este ejemplo se basa en la utilización del SORTF para imprimir una lista de jugadores en orden decreciente de puntuaciones. No hay más de 999 jugadores y sus puntuaciones o tiempos pueden representarse por un número de menos de seis dígitos.

Los datos forman un conjunto numérico a(); los puntos del jugador *n* están en el elemento a(n). Si ordenamos a(), tendremos los elementos por orden de valor numérico pero perderemos la identificación del jugador a quien corresponde cada uno.

Si programamos en BASIC:

```
FOR n = 1 TO...
LET a(n) = 1000* a(n) + n
NEXT n
```

entonces cada elemento contendrá ambas partes de la información. Los tres últimos dígitos identificarán al jugador y los restantes a su puntuación. Nótese que, debido a la manera en que el Spectrum trata a los números en coma flotante, el valor aparente de a(n) podría no ser mayor que 1.000.000.000. Podemos escribir ahora:

```
LET t = USR SORTF
REM al t:
```

y el conjunto será ordenado con las puntuaciones más altas al principio y el número de jugador contenido en las últimas tres cifras.

El elemento *n* puede ser escrito con:

```
PRINT INT (a(n) 1000); INT (a(n)-1000 * INT (a(n)/1000))
```

Queda aun otra consideración. Algunos elementos de a() pueden ser almacenados internamente en la forma entera, lo cual desconcertará a SORTF. Antes de utilizar la rutina, cada elemento debe estar en la forma de coma flotante, y la mejor manera de asegurar esto es empleando algo parecido a esto:

```
LET a(i) = a(i) + 65537 - 65537
```

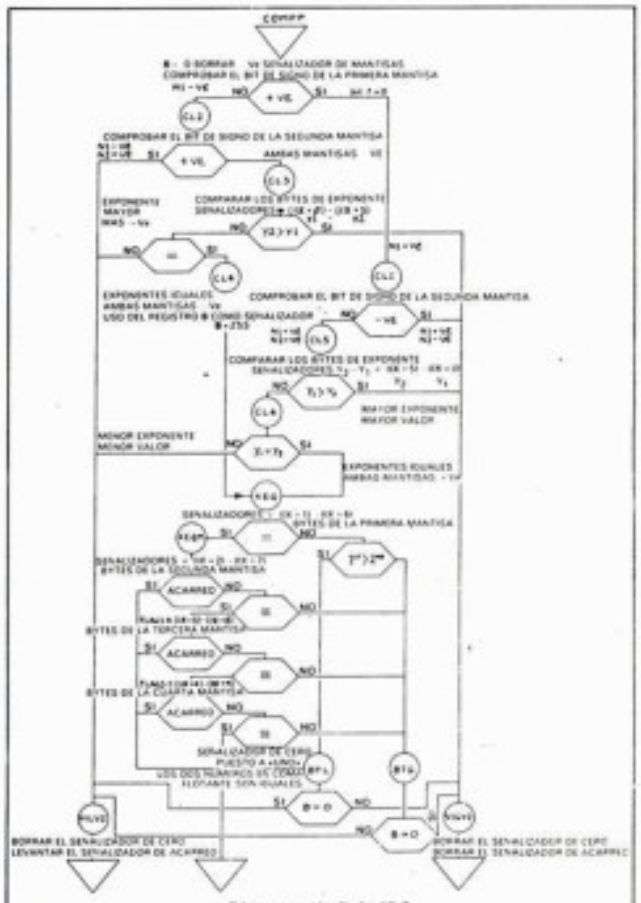



Diagrama de flujo 10.2

Para una lista de *m* entradas, el programa podria ser:

```
FOR n = 1 TO m
LET a(n) = 1000 * a(n) + n + 100 000
NEXT n
LET l = USR SORTF
REM a( ):
FOR n = 1 TO m
LET a(n) = a(n) - 100 000
PRINT INT(a(n)/1000); INT(a(n) - 1000*INT(a(n)/1000))
NEXT n
```

el cual escribirá la puntuación seguida del número de jugador. Donde haya dos jugadores con la misma puntuación, éstos estarán ordenados en forma decreciente por sus números de jugador.

Todo lo que se necesita es introducir los datos en a() para empezar y la rutina hará el resto en un abrir y cerrar de ojos.

Capítulo 11

EL TRASPASO DE OTROS PARAMETROS

Hasta aqui tenemos solamente unas cuantas rutinas que pueden ser llamadas desde el BASIC. Es relativamente fácil ensamblarlas de forma separada (con su propia dirección de carga y sus propias copias de las subrutinas comunes), procurando llamar correctamente a cada una cuando sea preciso. Las desventajas de este sistema son evidentes cuando tienen muchas subrutinas en común y èstas se multiplican, por tanto, innecesariamente.

La solución que yo he adoptado se muestra en el *listado 11.1*. Todas las rutinas, etc., están ensambladas juntas (todas las que se necesiten) y son llamadas a través de secuencias de código idènticas, que son todas de la misma longitud.

Primeramente se encuentra el almacenamiento del puntero de pila SP para facilitar el regreso a la linea 2 en caso de error; después, el almacenamiento de todos los registros utilizados, la llamada especifica a cada rutina (DRAWL, MAP$, etc.) y finalmente el salto a la etiqueta común de retorno TRAP$, donde los registros son restaurados y se efectúa el retorno RET al BASIC.

Listado 11.1

```
0005        ORG   60000
0010        LD    (ERROR),SP
0015        PUSH  AF
0020        PUSH  BC
0025        PUSH  DE
0030        PUSH  HL
0035        PUSH  IX
0040        CALL  DRAWL
```

```
0045        JP    TRAP#
0050        LD    (ERROR),SP
0055        PUSH  AF
0060        PUSH  BC
0065        PUSH  DE
0070        PUSH  HL
0075        PUSH  IX
0080        CALL  SATTR
0085        JP    TRAP#
0090        LD    (ERROR),SP
0095        PUSH  AF
0100        PUSH  BC
0105        PUSH  DE
0110        PUSH  HL
0115        PUSH  IX
0120        CALL  BLOCK
0125        JP    TRAP#
0130        LD    (ERROR),SP
0135        PUSH  AF
0140        PUSH  BC
0145        PUSH  DE
0150        PUSH  HL
0155        PUSH  IX
0160        CALL  SORTF
0165        JP    TRAP#
0170        LD    (ERROR),SP
0175        PUSH  AF
0180        PUSH  BC
0185        PUSH  DE
0190        PUSH  HL
0195        PUSH  IX
0200        CALL  GCELL
0205        JP    TRAP#
0210        LD    (ERROR),SP
0215        PUSH  AF
0220        PUSH  BC
0225        PUSH  DE
0230        PUSH  HL
0235        PUSH  IX
0240        CALL  MAP#
0245        JP    TRAP#
```



```
0250        LD    (ERROR),SP
0255        PUSH  AF
0260        PUSH  BC
0265        PUSH  DE
0270        PUSH  HL
0275        PUSH  IX
0280        CALL  IVERT
0285        JP    TRAP#
0290        LD    (ERROR),SP
0295        PUSH  AF
0300        PUSH  BC
0305        PUSH  DE
0310        PUSH  HL
0315        PUSH  IX
```

Estas entradas comunes son todas de 16 bytes de longitud y las rutinas pueden llamarse mediante un desplazamiento respecto a la dirección de origen:

DRAWL en USR + 0
SATTR en USR + 16
BLOCK en USR + 32

y así sucesivamente. Un programa BASIC (o solamente su inicio) podría parecerse al *listado 11.2*. Esto tiene la ventaja de que si hay que cambiar la dirección de carga de la rutina, solamente es necesario alterar la línea 10 del programa y que, después de la preparación inicial, las rutinas pueden ser llamadas por sus mnemónicos en vez de hacerlo por los valores numéricos. (Las rutinas SATTR y DRAWL se describen en los capítulos 13 y 14.)

Listado 11.2

```
 1 LET error=0: GO TO 10
 2 PRINT "ERROR =";error: STOP
10 LET base=60000
11 LET drawl=base+0
12 LET sattr=base+16
13 LET block=base+32
14 LET sortf=base+48
15 LET gcell=base+64
16 LET   map=base+80
17 LET ivert=base+96
```

```
 18 LET movec=base+112
 19 LET svert=base+126
 20  LET drawa=base+144
 21 LET demo1=base+160
 22 LET demo2=base+176
 25 GO SUB 70: PAUSE 200: GO SUB 80: GO SUB
  200: GO SUB 300: GO SUB 400: GO SUB
9000: GO TO 21
 30 LET b=250
 41 DIM k$(b,2)
 42 FOR x=1 TO b
 43 LET k$(x,1)= CHR$ 255
 44 LET k$(x,2)= CHR$ 255
 45 NEXT x
 50 LET k=0
 51 LET o1=0
 53 LET l= USR movec
 54 REM LEER POSICION DEL CURSOR
 56 PRINT  AT 0,0;"          ": PRINT  AT
 0,0;l: POKE 23560,255
 57 IF l=o1 THEN  LET l=65535
 58  LET k=k+1
 59  LET k$(k,2)= CHR$  INT (l/256)
 60  LET k$(k,1)= CHR$  INT (l-256*( INT
 (l/256)))
 61 IF k=1 THEN  GO TO 58
 62 LET m= USR drawa
 63 REM k$();
 64 LET o1=l
 65 PRINT  AT 0,6;k
 66 GO TO 53
 70 LET l= USR sattr
 71 REM :0,0,15,11,8,
 72 LET l= USR sattr
 73 REM :16,12,31,23,16,
 74 LET l= USR sattr
 75  REM :24,0,31,6,24,
 76 RETURN
 80 LET l= USR map
 81 RETURN
100 LOAD "" CODE : LOAD "" CODE : GO TO 1
```



```
200 DIM a(44)
201 FOR m=1 TO 44
202  LET a(m)= RND *10^( INT ((30* RND )-15))
203 NEXT m
204 GO SUB 220
205 LET l= USR sortf
206 REM a():
207 PAUSE 1
208 GO SUB 220
209 RETURN
220 CLS
221 FOR m=1 TO 44 STEP 2
222 PRINT a(m),a(m+1)
223 NEXT m
224 RETURN
300 PRINT  AT 3,5;"DEMOSTRACION DE COLOR."
301 FOR k=0 TO 255
302 LET l=demo1
303 REM k:0,0,14,7,
304 NEXT k
310  RETURN
400 DIM g(5)
401 FOR g=1 TO 500
402 LET g(5)= INT (255* RND )
403 LET g(3)= INT (31* RND )
404 LET g(4)= INT (23* RND )
405 LET g(1)= INT (g(3)* RND )
406 LET g(2)= INT (g(4)* RND )
407 LET l= USR demo2
408 NEXT g
409 RETURN
9000 POKE 23675,0: POKE 23676,150
9001 FOR x=0 TO 224 STEP 2
9002 LET l= USR gcell
9003 POKE 38400,x
9004 NEXT x: RETURN
```

Ahora deberia explicar algo: las sentencias REM en el listado 11.2. Antes, en el Capitulo 9, mostré cómo los NOMBRES de las variables podían ser traspasados pero dejamos a un lado (debido a que era todavía demasiado complicado) el traspaso de valores numéricos y

de cadenas. En el Capitulo 10 utilizamos un nombre de conjunto numérico traspasado para suministrar los punteros necesarios a la rutina SORTF. Ahora vamos a volver a las omisiones del Capitulo 10.

OPARS (Otros parámetros)

Estos parámetros deben ser compatibles con los nombres de parámetros recogidos por PCALL, es decir, que pueden estar presentes en la misma linea REM donde se encuentran los nombres de los parámetros. La manera más fácil de conseguir esto es dividiendo la lista de parámetros en dos partes: primero los nombres, terminados con un carácter de dos puntos, y luego los valores numéricos y cadenas. Puede darse el caso de que no existan nombres de parámetros pero, aun así, deberían mantenerse los dos puntos para señalar el inicio de la parte de valores y cadenas.

Las especificaciones para estos parámetros son:

– Cada elemento, incluido el último, debe terminar con una coma.

La linea REM debe terminar con un carácter ENTER.

– Los valores son enteros de 16 bits sin signo. (Sus valores pueden encontrarse en las variables VPARO, VPARO + 2, etc.)

– Las cadenas están delimitadas por comillas (») a cada extremo, pueden ser de cualquier longitud y deben terminar con una coma después de las comillas de cierre. No deben contener dobles comillas. La dirección del primer carácter de cada cadena puede hallarse en SPARO, SPARO + 2, etc.

– No se traspasa ningún dato acerca de las posiciones relativas de los valores y cadenas en la lista de parámetros; sólo se guarda constancia de sus posiciones relativas dentro de cada clase.

– Para permitir traspasar el 0 como un valor, se utiliza un byte subsidiario, SBITZ, y tiene los bits 7, 6, ... ajustados al valor 0 ó 1 dependiendo de si VPARO, VPARO + 2, etc., son válidos.

OPARS forzará los siguientes retornos por error:

10 no se ha encontrado el final de la linea REM
11 carácter que no es un dígito, en un número
12 demasiados parámetros (más de 6)
13 falsa lectura de un número
14 número mayor de 65535

Listado 11.3

```
4255 OPARS   LD    HL,SPARO
4260         LD    (SPZ),HL
4265         LD    HL,VPARO
4270         LD    (VPZ),HL
4275         LD    HL,0
4280         LD    (SPARO),HL
4285         LD    HL,SPARO
4290         LD    DE,SPARO+1
4295         LD    BC,SBITZ-SPARO+1
4300         LDIR
4305         LD    HL,(NXTLN)
4310         INC   HL
4315         INC   HL
4320         LD    (VPL+2),HL
4325 VPL     LD    BC,(VPL+2)
4330         INC   HL
4335         INC   HL
4340 GNB1    CALL  GETBY
4345         CP    13
4350         RET   Z
4355         CP    ":"
4360         JR    NZ,GNB1
4365 GNB2    CALL  GETBY
4370         CP    13
4375         RET   Z
4380         CP    " "
4385         JR    Z,GNB2
4390         CP    ","
4395         JR    Z,EOPAR
4400         CP    """"
4405         JP    Z,STSTR
4410         CP    "0"
4415         JP    M,ERX11
4420         CP    ":"
4425         JP    P,ERX11
4430         JR    Z,EOPAR
4435         SUB   "0"
4440         PUSH  HL
4445         PUSH  BC
```

```
4450            OR   A
4455            LD   HL,(NUMB)
4460            ADD  HL,HL
4465            JR   C,ERX14
4470            ADD  HL,HL
4475            JR   C,ERX14
4480            LD   BC,(NUMB)
4485            ADD  HL,BC
4490            JR   C,ERX14
4495            ADD  HL,HL
4500            JR   C,ERX14
4505            LD   B,0
4510            LD   C,A
4515            ADD  HL,BC
4520            JR   C,ERX14
4525            LD   (NUMB),HL
4530            LD   A,1
4535            LD   (NNR),A
4540            POP  BC
4545            POP  HL
4550            JR   GNB2
4555 ERX14      LD   DE,14
4560            CALL ERREX
4565 EOPAR      PUSH HL
4570            PUSH BC
4575            LD   A,(NNR)
4580            CP   0
4585            JP   Z,ERX13
4590            LD   A,(SBITZ)
4595            SRL  A
4600            SET  7,A
4605            LD   (SBITZ),A
4610            LD   HL,(NUMB)
4615            LD   BC,(VPZ)
4620            LD   (VPL2+1),BC
4625 VPL2       LD   (VPL2+1),HL
4630            INC  BC
4635            INC  BC
4640            LD   (VPZ),BC
4645            LD   HL,NUMB+2
4650            OR   A
```



```
4655            SBC  HL,BC
4660            JP   Z,ERX12
4665            LD   HL,0
4670            LD   (NUMB),HL
4675            LD   A,0
4680            LD   (NNR),A
4685            POP  BC
4690            POP  HL
4695            JP   GNB2
4700 STSTR      PUSH HL
4705            PUSH BC
4710            EX   DE,HL
4715            LD   BC,(SPZ)
4720            LD   HL,VPAR0
4725            OR   A
4730            SBC  HL,BC
4735            JR   Z,ERX12
4740            EX   DE,HL
4745            LD   (VPL3+1),BC
4750 VPL3       LD   (VPL3+1),HL
4755            INC  BC
4760            INC  BC
4765            LD   (SPZ),BC
4770            POP  BC
4775            POP  HL
4780 RFORC      CALL GETBY
4785            CP   """"
4790            JR   NZ,RFORC
4795 GNB3       CALL GETBY
4800            CP   ","
4805            JP   Z,GNB2
4810            CP   13
4815            RET  Z
4820            JR   GNB3
4825 GETBY      DEC  BC
4830            BIT  7,B
4835            JR   NZ,ERX10
4840            LD   A,(HL)
4845            INC  HL
4850            RET
4855 ERX10      LD   DE,10
```

```
ERXAA   CALL ERREX
ERX11   LD   DE,11
        JR   ERXAA
ERX12   LD   DE,12
        JR   ERXAA
ERX13   LD   DE,13
        JR   ERXAA
VPZ     DEFW 0
SPZ     DEFW 0
SPARO   DEFW 0
        DEFW 0
        DEFW 0
        DEFW 0
        DEFW 0
        DEFW 0
VPARO   DEFW 0
        DEFW 0
        DEFW 0
        DEFW 0
        DEFW 0
        DEFW 0
NUMB    DEFW 0
NNR     DEFB 0
SBITZ   DEFB 0
        DEFW 0
```

Operación

Como que la rutina puede llamarse muchas veces, todo el espacio de trabajo se borra primeramente y los punteros— SPZ para cadenas y VPZ para valores — se ajustan para señalar al inicio de las respectivas listas, SPARO y VPARO.

En VPL, el registro BC se carga con el número de caracteres en la línea de parámetros siguiente a la llamada USR ... y HL se ajusta para señalar al primer byte. GETBY lee los bytes en forma secuencial utilizando HL y decrementando BC (con error 10 si BC se vuelve negativo) el cual está preservado para esta utilización. El carácter leído se almacena en el registro A.

En GNB1 se leen los caracteres hasta que se encuentra un carácter 13 (ENTER) o bien un carácter de dos puntos. Los dos puntos

Diagrama de flujo 11.1

marcan el final de la parte de nombres de parámetros, la cual puede estar vacía.

En GNB2 se analizan los caracteres que se encuentran después de los dos puntos. Un carácter 13 termina la rutina. Los espacios son ignorados. Una coma se reconoce como final de parámetro (salto a EOPAR), y las comillas se reconocen como el inicio de una cadena que debe tratarse en STSTR. Todo lo que pueda quedar debe ser un dígito decimal o bien un error.

Números

Los códigos ASCII de los números están dispuestos en forma secuencial desde el 48 decimal para el «0» hasta el 58 dec. para el 9 y los dos puntos tienen el código ASCII 59 en decimal. Restándoles el código del 0 se obtiene una representación binaria válida del dígito obtenido.

Los registros HL y BC se almacenan para la siguiente utilización en GETBY y HL es cargado con NUMB, que contiene el resultado parcial de la evaluación del valor (o cero). HL se multiplica por 10 mediante el desplazamiento y suma y después se le suma A para obtener un nuevo resultado parcial el cual se guarda nuevamente en NUMB. En cada paso se comprueba HL por si se produce un desbordamiento («overflow») de su valor y se genera un error 10 si es preciso. NNR se ajusta a un valor no nulo para indicar que se está leyendo un número y se restauran los registros HL y BC, dispuestos para leer el siguiente byte de entrada.

Fin de parámetro (EOPAR)

Si NNR tiene un valor nulo, una condición de error (doble coma o un valor perdido) provoca la aparición del mensaje de error 13. En otro caso habrá sido leído un número válido y se podrá poner a «1» un nuevo bit de SBITZ. Si el número era cero, la entrada VPAR debería ser también cero. Una entrada no nula no puede utilizarse como comprobación para la presencia de entrada, como se hace en SPAR para las cadenas, ya que 0 es el inicio de la memoria ROM. VPL2 es una instrucción computada que carga HL en la lista VPAR y después se incrementa VPZ en 2 para señalar al elemento de los dos bytes siguientes. Si señala a NUMB + 2, la tabla se ha desbordado y se genera un error 12. NUMB y NNR se borran para quedar listos para el siguiente parámetro de valor.

Nota: La secuencia entre las etiquetas VPZ y SBITZ no debe ser alterada aunque se cambie el número de elementos en las listas VPAR y SPAR.

Inicio de cadenas (STSTR)

HL señala al byte que se encuentra inmediatamente después de las comillas que ha sido leído por GETBY. Se guardan HL y BC, y HL (la dirección del primer carácter de la cadena) se guarda en DE; OR A pone a cero cualquier señalizador de acarreo y se compara SPZ con VPARO, el cual indica el final de la lista SPAR. Si existen demasiados parámetros de cadena, se genera de nuevo un error 12. VPL3 es una instrucción de carga computada del valor restaurado de HL (desde DE) en la tabla de direcciones de las cadenas.

Una vez que la tabla de direcciones de las cadenas se ha cargado, RFORC lee la cadena para localizar las comillas del final y después la coma o bien el carácter 13.

Sinopsis

OPARS permite traspasar constantes, valores enteros y cadenas desde un programa BASIC a una rutina en código máquina. Estos parámetros deben colocarse en una línea REM después de un carácter de dos puntos.

Capítulo 12

BORRADO DE BLOQUES DE BASIC

Si se desea eliminar un conjunto de líneas de un programa en BASIC, porque hayan quedado obsoletas, por ejemplo, habrá que teclear normalmente cada número de línea por separado lo que puede hacer perder mucho tiempo. Muchos otros microordenadores poseen un comando DELETE a,b o similar, que elimina de una vez las líneas desde la a hasta la b.

La siguiente rutina usa OPARS para borrar cualquier cantidad de líneas de programa. Es aconsejable consultar al Capítulo 24 del manual del Spectrum al seguir la descripción de la rutina. Esta precisa de dos parámetros de valor —ambos números de línea— y borra desde la primera hasta la segunda (pero sin incluirla). La técnica consiste en el borrado de cada línea individualmente seguido del ajuste de VARS. El sistema BASIC debe ser preparado con CLEAR tanto antes como especialmente después de utilizar la rutina.

Primeramente vamos a ver algunas subrutinas para recoger las líneas individuales y examinarlas (véase *Diagrama de Flujo 12.1*). Nótese que existen varias formas de entrar en un bloque común de código.

SUPLN («Set Up LiNe pointers»)

SUPLN ajusta los punteros de las líneas de programa; es utilizada por las otras rutinas para señalar a la primera línea del programa BASIC. Esta y las otras eliminan los contenidos de los registros a la entrada, y las condiciones de salida son las siguientes:

HL	contiene el número de la (nueva) línea
BC	contiene la longitud en bytes de la línea de datos
DE	señala al primer carácter de la línea
Señalizador Z	está a «uno» si no hay más datos

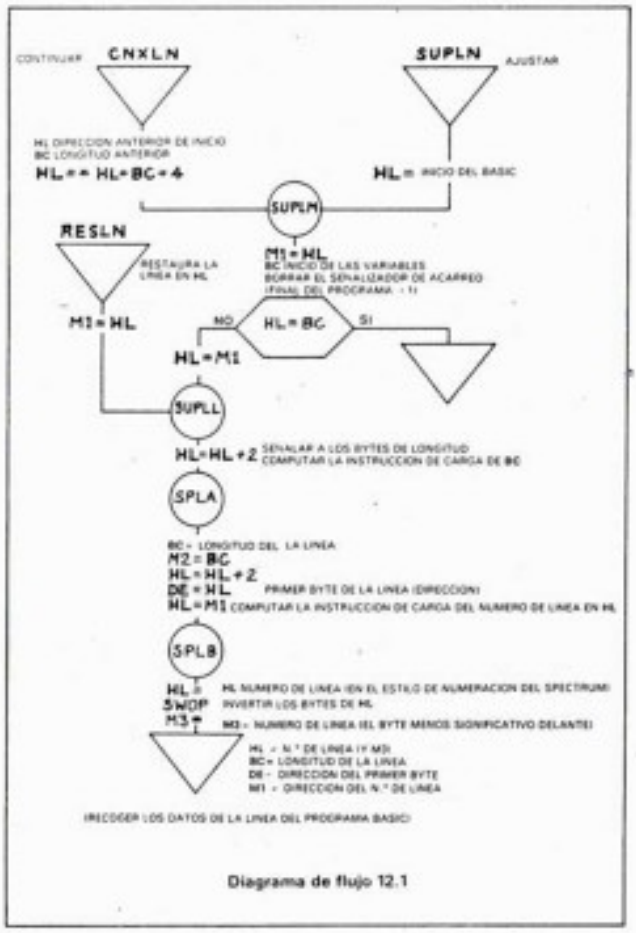


Diagrama de flujo 12.1

Las variables M1, M2, M3, M4 y M5 se utilizan de la forma siguiente:

M1	dirección del primer byte del número de línea
M2	longitud de esta línea en bytes (= BC)
M3	número de esta línea (= HL)
M4-M5	almacenamiento temporal mientras se borra una línea

Listado 12.1

```
4985 SUPLN  LD   HL,(PROG#)
4990 SUPLM  LD   (M1),HL
4995        LD   BC,(VARSS)
5000        OR   A
5005        SBC  HL,BC
5010        RET  Z
5015        LD   HL,(M1)
5020 SUPLL  INC  HL
5025        INC  HL
5030        LD   (SPLA+2),HL
5035 SPLA   LD   BC,(SPLA+2)
5040        LD   (M2),BC
5045        INC  HL
5050        INC  HL
5055        EX   DE,HL
5060        LD   HL,(M1)
5065        LD   (SPLB+1),HL
5070 SPLB   LD   HL,(SPLB+1)
5075        LD   A,H
5080        LD   H,L
5085        LD   L,A
5090        LD   (M3),HL
5095        RET
5100 CNXLN  LD   BC,(M2)
5105        LD   HL,(M1)
5110        ADD  HL,BC
5115        INC  HL
5120        INC  HL
5125        INC  HL
5130        INC  HL
5135        JR   SUPLM
```

```
5140 RESLN  LD    (M1),HL
5145        JR    SUPLL
5150 M1     DEFW  0
5155 M2     DEFW  0
5160 M3     DEFW  0
5165 M4     DEFW  0
5170 M5     DEFW  0
5175 PROG#  EQU   23635
```

CNXLN («Continue with NeXt LiNe»)

(Continúa con la línea siguiente.) Esta subrutina utiliza los registros de forma parecida a SUPLN pero para la siguiente línea de programa. Ajusta HL no a PROG, como hace SUPLN, sino a M1 + M2 + 4, que es el primer byte de la línea siguiente.

RESLN («REStore LiNe»)

(Restaura la línea.) Después de que una línea haya sido borrada, en el lugar donde empezaba aquélla se encuentra ahora el comienzo de la siguiente línea no borrada. RESLN ajusta los registros y las posiciones de memoria de acuerdo con esta nueva línea.

OPERACION DE SUPLN

A la entrada, HL contiene la localización de una línea (inicialmente la primera). Esta dirección se almacena en M1 y se compara con el valor de VARS. Si HL ha alcanzado ya VARS, significa que ya no hay más líneas y se efectúa una instrucción RET Z.

En SUPLL se incrementa HL en dos unidades para señalar a los bytes de longitud de línea, y este valor se almacena en SPLA + 2, que es la segunda mitad de la siguiente instrucción. La instrucción computada en SPLA carga BC con la longitud de la línea actual y la guarda en M2. La instrucción computada en SPLB carga entonces HL con el número de línea, el cual se encuentra en posición invertida y, por tanto, se invierten entre sí los registros H y L y se almacenan en M3, ejecutándose después el retorno. En todas las circunstancias normales, el señalizador Z estará a cero ya que no hay ninguna instrucción que afecte a los señalizadores, aparte de la comprobación

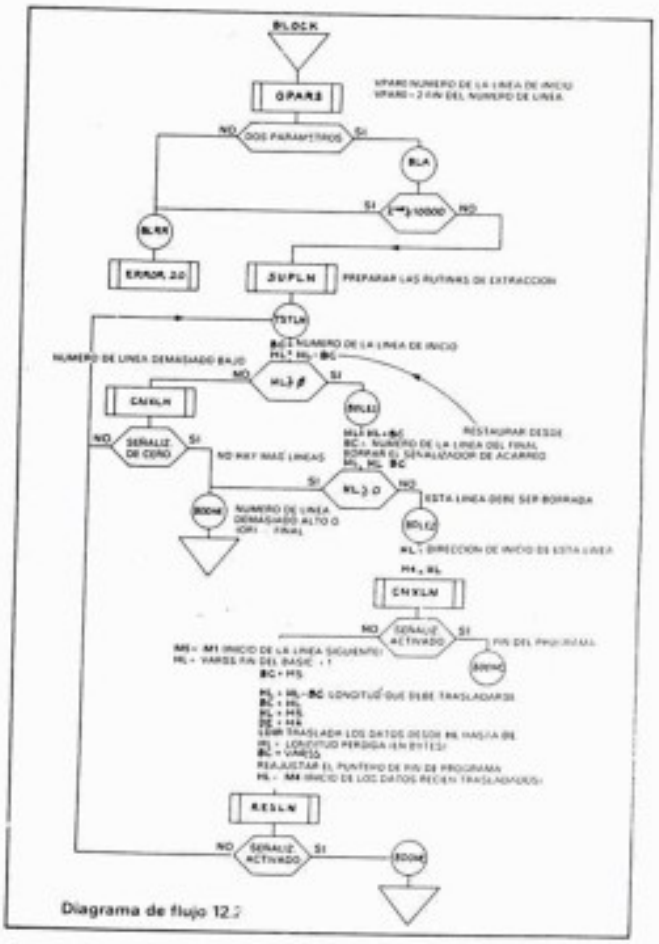


Diagrama de flujo 12.2

de SBC, después de SUPLM. Téngase en cuenta que al entrar en RESLN, el señalizador Z tiene valor cero.

OPERACION DE BLOCK
(Véase Diagrama de flujo 12.2)

BLOCK precisa de dos parámetros, y su llamada se efectúa de la forma siguiente:

```
LET L = USR...
REM : 174, 8234,
```

Si el segundo parámetro es menor que el primero, la rutina no hace nada. Se llama a OPARS para leer los dos parámetros, los cuales serán almacenados en VPARO y VPARO + 2 como números de 16 bits.

Se comprueba SBITZ para asegurar que sólo hay dos parámetros (si produce error 20 en cualquier otro caso), y el valor del segundo parámetro se comprueba también para asegurarse de su validez (debe ser menor de 10000). Se llama entonces a SUPLN para señalar a la primera línea BASIC y en TSTLN se compara el número de línea con el valor del primer parámetro. Si el valor es demasiado bajo, se llama a CNXLN para que localice la línea siguiente, y el proceso se repite mientras queden líneas para ser examinadas. Si el número de línea es igual o mayor que el primer parámetro, se produce entonces un salto hacia la rutina BDLE1.

Listado 12.2

```
5180 BLOCK  CALL OPARS
5185        LD   A,(SBITZ)
5190        XOR  128+64
5195        JR   Z,BLA
5200 BLRR   LD   DE,20
5205        CALL ERREX
5210 BLA    LD   HL,(VPARO+2)
5215        LD   BC,10000
5220        OR   A
5225        SBC  HL,BC
5230        JP   P,BLRR
5235        CALL SUPLN
```



```
5240 TSTLN  LD   BC,(VPARO)
5245        OR   A
5250        SBC  HL,BC
5255        JP   P,BDLE1
5260        CALL CNXLN
5265        JR   Z,BDONE
5270        JR   TSTLN
5275 BDLE1  ADD  HL,BC
5280        LD   BC,(VPARO+2)
5285        OR   A
5290        SBC  HL,BC
5295        JP   P,BDONE
5300 BDLE2  LD   HL,(M1)
5305        LD   (M4),HL
5310        CALL CNXLN
5315        JR   Z,BDONE
5320        LD   HL,(M1)
5325        LD   (M5),HL
5330        LD   HL,(VARSS)
5335        LD   BC,(M5)
5340        OR   A
5345        SBC  HL,BC
5350        PUSH HL
5355        POP  BC
5360        LD   HL,(M5)
5365        LD   DE,(M4)
5370        LDIR
5375        OR   A
5380        LD   HL,(M5)
5385        LD   BC,(M4)
5390        SBC  HL,BC
5395        PUSH HL
5400        POP  BC
5405        LD   HL,(VARSS)
5410        SBC  HL,BC
5415        LD   (VARSS),HL
5420        LD   HL,(M4)
5425        CALL RESLN
5430        JR   NZ,TSTLN
5435 BDONE  RET
```

BDLE1

El número de línea se compara con el valor del segundo parámetro, la línea límite superior. Si es menor que este límite, la línea debe ser eliminada mediante BDLE2; en otro caso, la rutina sale en BDONE hacia el cruel y frío mundo del programa BASIC acortado.

BDLE2

La línea debe ser borrada. La dirección de inicio se guarda en M4, y se llama a CNXLN para determinar la dirección de inicio y la presencia de la línea siguiente. BLOCK, específicamente, NO borrará la última línea del programa BASIC.

Se carga BC con el número de bytes que deben ser retenidos (desde el inicio de la línea recogida por CNXLN hasta la dirección en VARS) y se preparan DE y HL para que la instrucción LDIR desplace todo el conjunto de forma que cubra a la línea eliminada.

VARS se reduce entonces según la longitud de la línea eliminada y se llama a RESLN (con el señalizador Z a cero).

El proceso se repite en TSTLN, donde se comprueba la siguiente línea y se borra si es necesario.

Capítulo 13

AREA DE ATRIBUTOS

El área de atributos controla los colores de tinta y papel («INK» y «PAPER») y las características de brillo y parpadeo («BRIGHT» y «FLASH») de cada carácter de la pantalla. Estos atributos están dispuestos en forma secuencial a partir de la posición 22528, en forma de 24 filas de 32 columnas. Esta rutina permite alterar todos o algunos de los atributos en un área rectangular especificando el límite superior izquierdo y el inferior derecho del área deseada, junto con el byte de atributo requerido.

La llamada es de la forma:

```
LET L = USR...
REM : X1, Y1, X2, Y2, A,
```

El valor de las X debe estar comprendido entre 0 y 31, y el de las Y entre 0 y 23. El valor A (atributo) es un número decimal, según la *Figura 13.1*, que define las características que van a ser presentadas en pantalla. Recuérdese que es posible ocultar el contenido de la pantalla ajustando al mismo color la tinta y el papel, revelándose posteriormente el contenido al cambiar uno de los dos colores.

En la rutina pueden generarse dos errores:

30 límite superior izquierdo más bajo o más a la derecha que el límite inferior derecho.

31 alguno de los recuadros especificados se encuentra fuera del área de atributos.

OPERACIÓN (Ver Diagrama de Flujo 13.1)

OPARS recoge los valores que se suponen presentes, y se calcula STRTA como la dirección del primer byte de atributos que debe ser cargado.

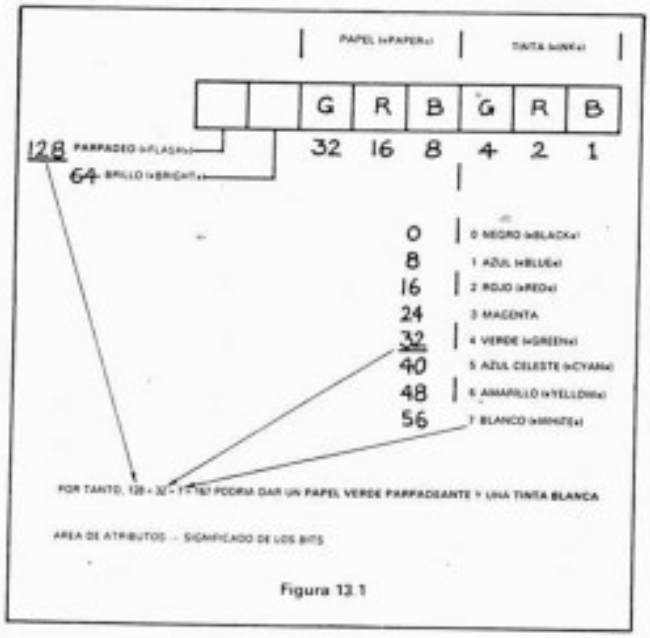


Figura 13.1

CDIFF contiene la diferencia de + 1 en las columnas (valores X) especificados y RDIFF la diferencia de + 1 en las filas. Si los valores de fila o de columna son los mismos, se podrá tratar una sola fila o columna. Cuando se efectúen varias llamadas, recuerde que si el límite inferior derecho de una coincide con el límite superior izquierdo de otra, se producirá un solapado entre ambos, siendo el anterior carácter sobreescrito por el último.

Una vez que RDIFF y CDIFF han sido ajustados, el bucle doble de la rutina ATTRL F 13 escribe las RDIFF filas de CDIFF atributos. Cada fila de atributos comienza 32 bytes después del inicio de la fila anterior y no hay que tratar con las complicaciones del trazado «pixels» (PLOT).

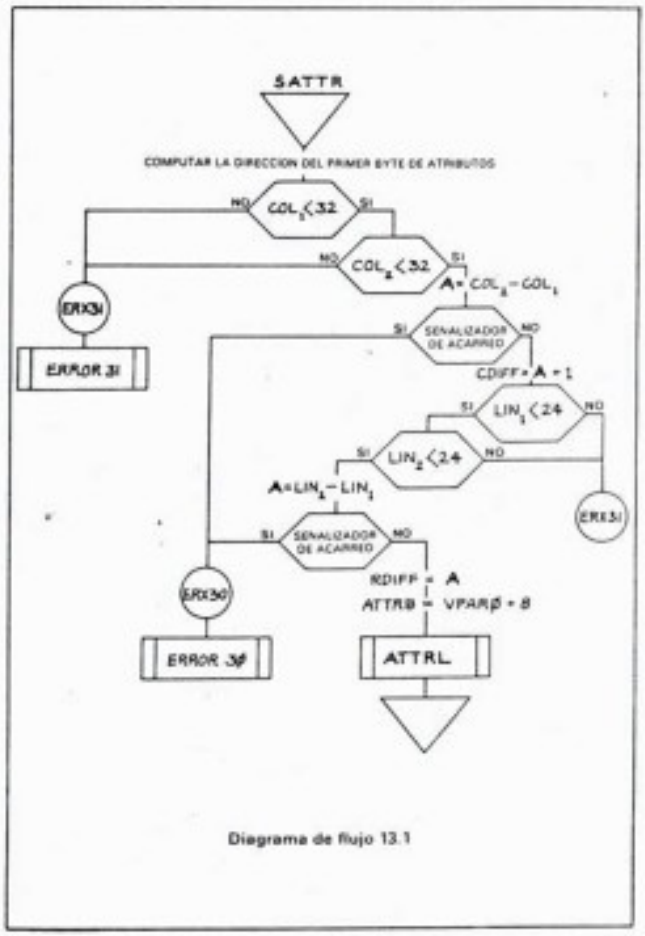


Diagrama de flujo 13.1

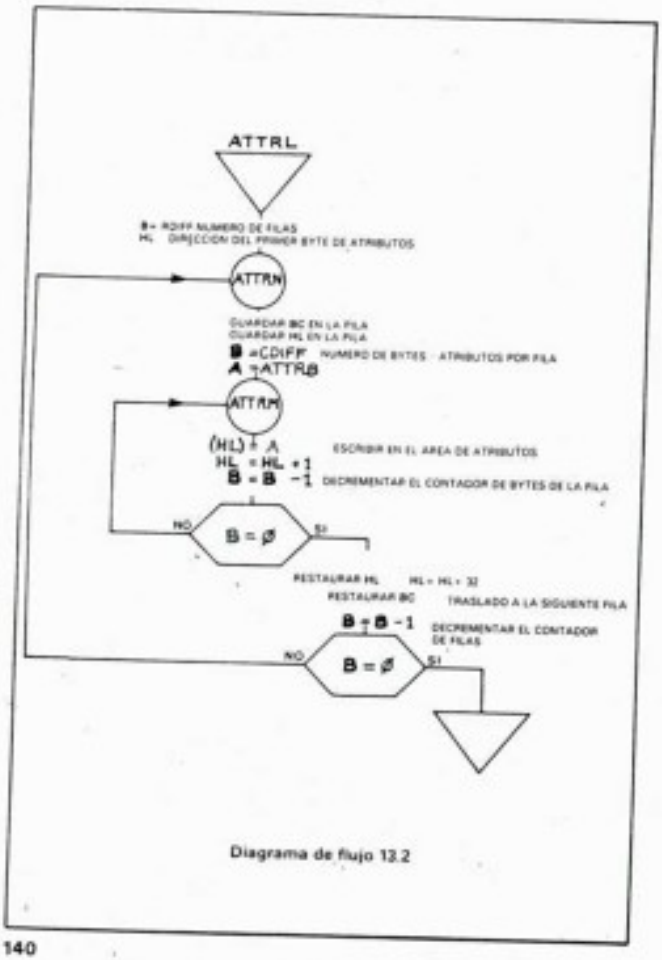


Diagrama de flujo 13.2

Listado 13.1

```
5510 SATTR  CALL  OPARS
5515        LD    HL,(VPAR0+2)
5520        ADD   HL,HL
5525        ADD   HL,HL
5530        ADD   HL,HL
5535        ADD   HL,HL
5540        ADD   HL,HL
5545        LD    BC,(VPAR0)
5550        ADD   HL,BC
5555        LD    BC,16384+6144
5560        ADD   HL,BC
5565        LD    (STRTA),HL
5570        LD    A,(VPAR0)
5575        CP    32
5580        JP    P,ERX31
5585        LD    B,A
5590        LD    A,(VPAR0+4)
5595        CP    32
5600        JP    P,ERX31
5605        SUB   B
5610        JR    C,ERX30
5615        INC   A
5620        LD    (CDIFF),A
5625        LD    A,(VPAR0+2)
5630        CP    24
5635        JP    P,ERX31
5640        LD    B,A
5645        LD    A,(VPAR0+6)
5650        CP    24
5655        JP    P,ERX31
5660        SUB   B
5665        JR    C,ERX30
5670        INC   A
5675        LD    (RDIFF),A
5680        LD    A,(VPAR0+8)
5685        LD    (ATTRB),A
5690        CALL  ATTRL
5695        RET
5700 ATTRB  DEFB  0
```

```
5705 CDIFF   DEFB 0
5710 RDIFF   DEFB 0
5715 STRTA   DEFW 0
5720 ERX30   LD   DE,30
5725         CALL ERREX
5730 ERX31   LD   DE,31
5735         CALL ERREX
```

Listado 13.2

```
5740 ATTRL   LD   A,(RDIFF)
5745         LD   B,A
5750         LD   HL,(STRTA)
5755 ATTRN   PUSH BC
5760         PUSH HL
5765         LD   A,(CDIFF)
5770         LD   B,A
5775         LD   A,(ATTRB)
5780 ATTRM   LD   (HL),A
5785         INC  HL
5790         DJNZ ATTRM
5795         POP  HL
5800         LD   BC,32
5805         ADD  HL,BC
5810         POP  BC
5815         DJNZ ATTRN
5820         RET
```


Capitulo 14

GRAFICOS EN ALTA RESOLUCION

El Spectrum tiene una resolución en pantalla de 256 «pixels» horizontales por 192 verticales. En este capitulo se presentan rutinas para trazar líneas y mover un cursor a través de ellas, así como un programa elemental de dibujo.

La única forma de trazar una línea entre dos puntos de la pantalla del Spectrum es dibujar, punto por punto, mediante PLOT, todos los puntos posibles en la línea entre X_1, Y_1 y X_2, Y_2, efectuándolo preferiblemente de una forma rápida.

Una manera de hacer esto que produce unos resultados razonables es la siguiente: encontrando incrementos DX y DY, no necesariamente enteros o positivos en X e Y que puedan ser sumados repetidamente a X_1, Y_1 para que la línea formada se dirija a X_2, Y_2. Esto es, en principio, lo que sucede cuando se dibuja una línea con una regla en un papel cuadriculado.

Los problemas aparecen ahora. ¿Cómo vamos a manejar las fracciones si hasta ahora sólo hemos tratado con números enteros? ¡No se asuste! La respuesta no está en los números en coma flotante sino en el sistema de escalado.

El escalado es una técnica muy utilizada en la programación en lenguaje máquina para manejar valores distintos al byte o palabra normales de la máquina. A título de ejemplo, tomaremos los puntos X e Y de la pantalla como números de 16 bits. El byte más significativo representará los puntos reales que pueden ser trazados y el byte menos significativo representará las partes fraccionales que no pueden trazarse.

Tomamos las diferencias aritméticas (con signo) entre las X y entre las Y y dividimos cada una de ellas por 256 (mediante el cambio de signo del byte) para generar las diferencias DX y DY. Esto funciona debido a que la máxima diferencia entre dos X, es de 255 pero

recordemos que hay que tratar a DX y DY como valores de 16 bits y tener en cuenta su signo en el byte más significativo. Para reducir el trabajo al trazado (PLOT), DX y DY se desplazan a la izquierda hasta que su dígito más significativo representa una cuarta parte de un punto. Cuanto más discontinua sea la línea resultante menos tiempo requerirá. La elección es suya y debería realizar pruebas modificando la rutina SDIFF que ajusta DX y DY antes de ser utilizados

Listado 14.1

```
5825 PLINE   CALL  OPARS
5830         LD    A,(VPAR0)
5835         LD    B,A
5840         LD    A,(VPAR0+2)
5845         LD    C,A
5850         LD    A,(VPAR0+4)
5855         LD    D,A
5860         LD    A,(VPAR0+6)
5865         LD    E,A
5870 XLINE   LD    (YO+1),BC
5875         LD    (XO),BC
5880         LD    (Y1+1),DE
5885         LD    (X1),DE
5890         LD    A,0
5895         LD    (XO),A
5900         LD    (X1),A
5905         LD    (YO),A
5910         LD    (Y1),A
5915         LD    (DX),A
5920         LD    HL,(X1+1)
5925         LD    BC,(XO+1)
5930         OR    A
5935         SBC   HL,BC
5940         LD    (DX),HL
5945         LD    (OLDDX),HL
5950         LD    HL,(Y1+1)
5955         LD    BC,(YO+1)
5960         OR    A
5965         SBC   HL,BC
5970         LD    (DY),HL
5975         LD    (OLDDY),HL
5980         CALL  SDIFF
5985 NPOIN   LD    A,(YO+1)
5990         LD    B,A
5995         LD    D,A
6000         LD    A,(XO+1)
6005         LD    C,A
6010         LD    E,A
6015         CALL  PLOT
6020 INVPT   OR    (HL)
6025         LD    (HL),A
6030         CALL  LPOIN
6035         RET   Z
6040 GNXPT   LD    HL,(XO)
6045         LD    BC,(OLDDX)
6050         ADD   HL,BC
6055         LD    (XO),HL
6060         LD    HL,(YO)
6065         LD    BC,(OLDDY)
6070         ADD   HL,BC
6075         LD    (YO),HL
6080         LD    A,(YO+1)
6085         CP    D
6090         JR    NZ,NPOIN
6095         LD    A,(XO+1)
6100         CP    E
6105         JR    Z,GNXPT
6110         JR    NPOIN
6115 LPOIN   LD    A,(XO+1)
6120         LD    B,A
6125         LD    A,(X1+1)
6130         CP    B
6135         RET   NZ
6140         LD    A,(YO+1)
6145         LD    B,A
6150         LD    A,(Y1+1)
6155         CP    B
6160         RET
6165 YO      DEFW  0
6170 XO      DEFW  0
6175 Y1      DEFW  0
6180 X1      DEFW  0
```

XLINE

XLINE es otra entrada a la rutina utilizada por DRAWL (véase más adelante) que traza una serie de líneas. B, C, D, E se cargan en los bytes más significativos de X_0, Y_0, X_1 e Y_1, y los bytes menos significativos son borrados. Obsérvese cuidadosamente cómo se realiza el almacenamiento el cual *no debe alterarse* ya que, de otra forma, se precisarían más instrucciones.

X_1 y X_0 se cargan en los bytes «bajos» de HL y BC, los bytes «altos» son cero y DX es un valor de 16 bits con signo formado por $X_1 - X_0$. El byte «alto» es cero o bien todo «unos».

De forma similar, DY se forma a partir de $Y_1 - Y_2$. La subrutina SDIFF produce los valores de DX y DY tan grandes como sea posible pero no más de un cuarto de un elemento de imagen («pixel»), y coloca sus valores en OLDDX y OLDDY.

NPOIN

Aquí es donde se dibuja el punto siguiente. BC (y DE) se cargan con las coordenadas Y en B, X en C y se llama a PLOT. INVPT, que puede ser una instrucción OR o XOR, modifica el contenido de la memoria de pantalla (buffer). X_0 e Y_0, como números de 16 bits, se incrementan con los valores fraccionarios de OLDDX y OLDDY hasta que los nuevos X_0 o Y_0 difieran de los antiguos valores almacenados en DE.

Este nuevo punto calculado se dibuja y se repite el proceso hasta que el punto dibujado coincide con X_1, Y_1, donde la subrutina NPG retorna con el señalizador Z a «uno».

SDIFF

Todo esto está hecho en forma de remiendos pero puede ser mejorado. DX y DY pueden ser desplazados a la izquierda mientras sus bytes más significativos sean todos ceros o todos unos y desplazados después dos lugares a la derecha.

Ahora podemos ya dibujar una línea entre dos puntos. Probablemente no lo utilizará usted mucho ya que el próximo paso es aún más interesante.

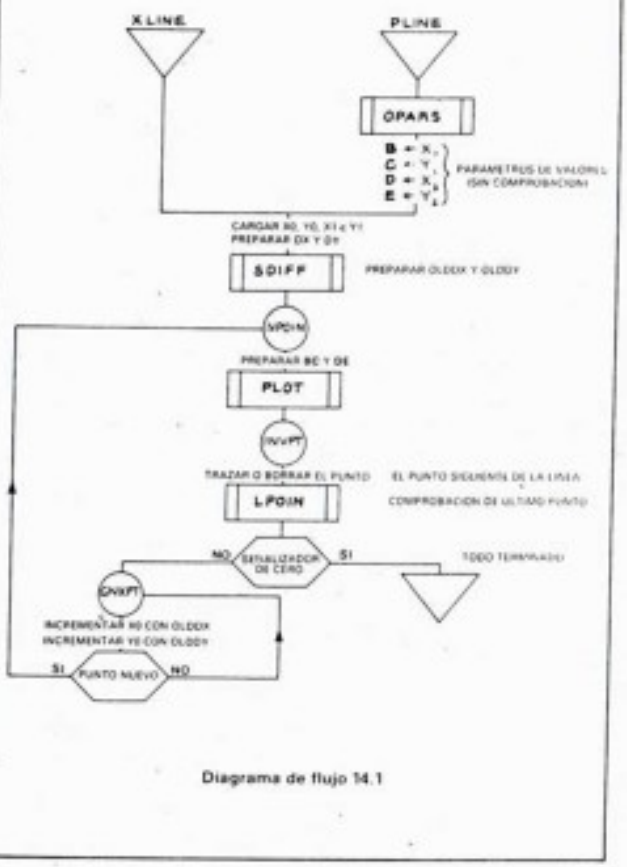


Diagrama de flujo 14.1

Listado 14.2

```
6350 DRAWL   CALL  OPARS
6355         LD    HL,(SPARO)
6360         CALL  GVAL8
6365         RET   C
6370         LD    B,A
6375         CALL  GVAL8
6380         RET   C
6385         LD    C,A
6390 DRNXP   CALL  GVAL8
6395         RET   C
6400         LD    D,A
6405         CALL  GVAL8
6410         RET   C
6415         LD    E,A
6420         PUSH  HL
6425         CALL  XLINE
6430         POP   HL
6435         PUSH  DE
6440         POP   BC
6445         JR    DRNXP
6450 GVAL8   PUSH  BC
6455         PUSH  DE
6460 NBY     LD    A,(HL)
6465         INC   HL
6470         CP    """"
6475         JR    Z,JRCX
6480         CP    ","
6485         JR    Z,JRVX
6490         SUB   "0"
6495         LD    B,A
6500         LD    A,(BYTEV)
6505         SLA   A
6510         SLA   A
6515         LD    C,A
6520         LD    A,(BYTEV)
6525         ADD   C
6530         SLA   A
6535         ADD   B
6540         LD    (BYTEV),A
```



```
6545         JR    NBY
6550 JRVX    LD    A,(BYTEV)
6555         LD    B,A
6560         LD    A,0
6565         LD    (BYTEV),A
6570         LD    A,B
6575         OR    A
6580         POP   DE
6585         POP   BC
6590         RET
6595 JRCX    SCF
6600         POP   DE
6605         POP   BC
6610         RET
6615 BYTEV   DEFB  0
```

DRAWL
Trazado de una lista de líneas

DRAWL tiene sólo un parámetro: una cadena cuyo contenido es una lista de dígitos y comas interpretados como pares X e Y que la rutina traza desde el par 1 al 2, desde éste al 3, y así sucesivamente hasta el final de la lista. Nótese que no se comprueba la validez de los valores, excepto cuando GVAL8 pasa solamente los 8 bytes menos significativos de cualquier valor que encuentre. Estas comprobaciones pueden ser insertadas si se desea.

OPARS recoge un parámetro de la cadena y GVAL8 recupera los valores de los bytes desde la cadena en una forma muy primitiva de cargar C, B, E, D con la llamada a XLINE para trazar la línea desde BC a DE.

DE se traspasa a BC y se carga DE con la siguiente posición del punto, trazándose a continuación la línea entre BC y DE. El proceso continúa hasta que GVAL8 sale con el señalizador de acarreo a uno, como consecuencia del fin de los datos de la cadena.

GVAL8

La entrada se produce con HL apuntando al parámetro de cadena. Se carga A con el siguiente carácter que se supone debe ser:

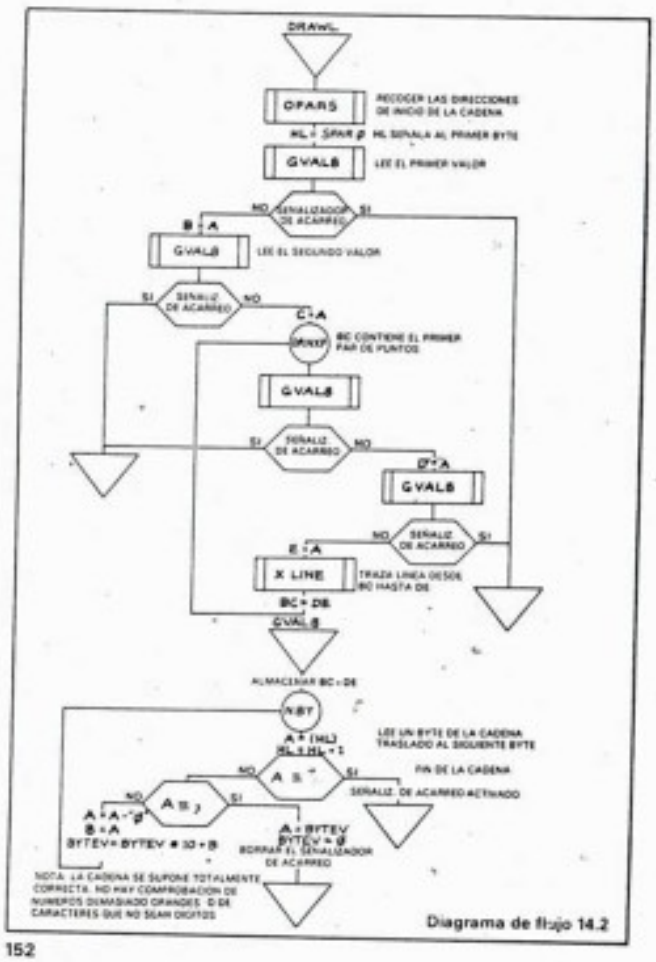


Diagrama de flujo 14.2

— Un carácter de comillas indicando el final de la cadena.
— Un carácter de coma, indicando el final de un valor.
— Un dígito. Los caracteres que no son dígitos no son rechazados sino macerados.

BYTEV: BYTe EValuado

Esto se efectúa desplazando y sumando para multiplicar por 10 y sumando después el valor binario del carácter que se supone un dígito. No hay comprobaciones y el proceso sigue hasta que se lee una coma.

Ahora que ya podemos trazar líneas, ¿qué hay del borrado de las mismas? Esto no es excesivamente difícil. Cambiando el OR (HL) en INVPT a XOR (HL), todo funcionará bien siempre y cuando sigamos los pasos con precisión. Debido a que hay, o debe haber, muchos puntos donde debe realizarse este cambio, la subrutina IVERT contiéne una lista de bytes que deben cambiarse. Cada llamada a IVERT cambia y vuelve a cambiar los mismos. Para aquellos que se pierdan, existe SVERT que prepara todas las opciones en OR para el dibujo.

Listado 14.3

```
6620 IVERT  LD   A,(INVPT)
6625        LD   B,A
6630        LD   A,(CHNGE)
6635        LD   (INVPT),A
6640        LD   (INVRX),A
6645        LD   A,B
6650        LD   (CHNGE),A
6655        RET
6660 CHNGE  XOR  (HL)
6665 SVERT  LD   A,(INVPT)
6670        LD   B,A
6675        LD   A,(XOROP)
6680        CP   B
6685        RET  NZ
6690        CALL IVERT
6695        RET
6700 XOROP  XOR  (HL)
```

MOVEC

Esta rutina de movimiento del cursor opera mediante el trazado y borrado de un rombo de puntos. Para un movimiento más rápido, debe incrementarse la posición del cursor en más de un elemento de imagen (un «pixel») o saltar cada vez un recuadro de carácter.

La base es una llamada a IFKEY que opera de la forma siguiente:

Las teclas 5, 6, 7 y 8 mueven el cursor en las direcciones obvias.

s («slow») conecta el movimiento lento.
f («fast») conecta el movimiento rápido.
x conecta el modo de paso a paso.
p hace salir la rutina y entrega la posición del cursor.

Todas estas teclas deben pulsarse en posición de minúsculas.

Una llamada LET L = USR ... asigna a L, cuando se pulsa la tecla «p», la posición del cursor, que puede ser descifrada por el programa BASIC. La pulsación prolongada de la tecla produce salidas repetidas de la misma posición. A la primera llamada, el cursor está posicionado cerca del centro de la pantalla pero las repetidas llamadas recuperan la última posición conocida del cursor.

Listado 14.4

```
6705 MOVEC  CALL SVERT
6710        CALL CURSR
6715        CALL IVERT
6720 MFAST  LD   A,1
6725        LD   (23561),A
6730        LD   (23562),A
6735 CIFKE  CALL IFKEY
6740        DEFB "8"
6745        JP   MRGHT
6750        DEFB "5"
6755        JP   MLEFT
6760        DEFB "7"
6765        JP   MUPUP
6770        DEFB "6"
6775        JP   MDOWN
6780        DEFB "p"
6785        JP   MEXIT
6790        DEFB "f"
6795        JP   MFAST
6800        DEFB "s"
6805        JP   MSLOW
6810        DEFB "x"
6815        JP   MSTEP
6820        NOP
6825 MSTEP  LD   A,255
6830        LD   (23561),A
6835 MFASU  LD   (23562),A
6840        JR   CIFKE
6845 MSLOW  LD   A,10
6850        JR   MFASU
6855 MRGHT  LD   A,(CURSX)
6860 MRL1   INC  A
6865        CP   253
6870        JR   Z,MRLA
6875        LD   (CURSX),A
6880        JR   CIFKP
6885 MLEFT  LD   A,(CURSX)
6890 MRLA   DEC  A
6895        CP   2
6900        JR   Z,MRL1
6905        LD   (CURSX),A
6910        JR   CIFKP
6915 MUPUP  LD   A,(CURSY)
6920 MUPL1  DEC  A
6925        CP   2
6930        JR   Z,MDWNA
6935        LD   (CURSY),A
6940        JR   CIFKP
6945 MDOWN  LD   A,(CURSY)
6950 MDWNA  INC  A
6955        CP   188
6960        JR   Z,MUPL1
6965        LD   (CURSY),A
6970        JR   CIFKP
6975 CURSX  DEFB 125
6980 CURSY  DEFB 88
6985 CIFKP  CALL CROSS
6990        CALL IVERT
```

```
6995            CALL  CURSR
7000            CALL  IVERT
7005            JR    CIFKE
7010 PLOA       DEFB  254
7015            DEFB  254
7020 PLOB       DEFB  254
7025            DEFB  +2
7030 PLOC       DEFB  +2
7035            DEFB  +2
7040 PLOD       DEFB  +2
7045            DEFB  254
7050 PLOAT      DEFW  0
7055 PLOBT      DEFW  0
7060 PLOCT      DEFW  0
7065 PLODT      DEFW  0
7070 CURSR      LD    BC,(CURSX)
7075            LD    HL,(PLOA)
7080            ADD   HL,BC
7085            LD    (PLOAT),HL
7090            LD    HL,(PLOB)
7095            ADD   HL,BC
7100            LD    (PLOBT),HL
7105            LD    HL,(PLOC)
7110            ADD   HL,BC
7115            LD    (PLOCT),HL
7120            LD    HL,(PLOD)
7125            ADD   HL,BC
7130            LD    (PLODT),HL
7135            CALL  CROSS
7140            RET
7145 CROSS      LD    BC,(PLODT)
7150            CALL  XPLOT
7155            LD    BC,(PLOBT)
7160            CALL  XPLOT
7165            LD    BC,(PLOAT)
7170            CALL  XPLOT
7175            LD    BC,(PLOCT)
7180            CALL  XPLOT
7185            RET
7190 XPLOT      CALL  PLOT
7195 INVRX      OR    (HL)
```



```
7200            LD    (HL),A
7205            RET
7210 MEXIT      CALL  SVERT
7215            POP   IX
7220            POP   HL
7225            POP   DE
7230            POP   BC
7235            LD    BC,(CURSX)
7240            LD    A,5
7245            LD    (23562),A
7250            LD    A,35
7255            LD    (23561),A
7260            JP    TRAT$
```

OPERACION DE MOVEC

Esta rutina es tan sencilla que no es necesario dibujar su diagrama de flujo sino que trabajaremos directamente sobre el listado.

SVERT ajusta la rutina de trazado a un estado conocido y después se ajustan también las variables REPDEL y REPPER a sus valores mínimos para conseguir el movimiento más rápido posible. La posición inicial del cursor se dibuja con una llamada a CURSR. Se llama entonces a IVERT para que la siguiente llamada borre la posición actual del cursor antes de dibujar la segunda posición. Esto produce un movimiento bastante regular.

La rutina IFKEY espera ahora a que se produzca la pulsación de una de las letras minúsculas del menú. Las teclas del cursor 5, 6, 7 y 8 provocan saltos a MLEFT, MRIGHT, MUPUP y MDOWN, donde los bytes de posición del cursor CURSX y CURSY se modifican adecuadamente y se impide que puedan salir fuera de la pantalla. Se borra la antigua posición y se traza la nueva antes de retornar a CIFKE para la siguiente pulsación de tecla.

Las teclas x, s y f cargan REPPER con el valor adecuado. Nótese que un elemento de imagen (un «píxel») cubre verticalmente tres líneas de exploración del televisor.

OTROS DETALLES

PLOA, PLOB, PLOC y PLOD definen los cuatro puntos del rombo respecto a la posición del cursor para que los puntos actuales puedan

definirse con la adición de CURSX, considerado con CURSY, como un valor de dos bytes para estos cuatro puntos. Estas adiciones dan como resultado los puntos PLOAT, PLOBT, PLOCT y PLODT que son luego trazados y borrados por CROSS y XPLOT (de acuerdo con el estado de INVRX, que es ajustado por IVERT o SVERT).

Cuando se pulsa la tecla «p», la rutina sale a través de MEXIT, la cual restaura los registros, excepto BC, en el que se coloca la posición del cursor. Como ya es normal en mis rutinas, las posiciones de las declaraciones de bytes o palabras son importantes.

DRAWA: Trazado de matriz

Con esta subrutina y MOVEC se puede construir un sencillo programa de dibujo como se indica en el *listado 14.5b*.

DRAWL busca sus datos como valores de puntos en una lista de parámetros localizada en una línea REM. DRAWA es una variación sobre el mismo tema pero, en este caso, los datos se encuentran en una matriz bidimensional que puede ser definida como:

```
DIM  ?$ ( ... , 2)
```

donde «?» significa cualquier referencia válida para una matriz y «...» es tan grande como se precise. El par de caracteres ?$ (p,1) y ?$ (p,2) contiene los valores X e Y para el trazado del punto *p* como valores de un byte. Si el valor Y sale fuera de la pantalla, el punto se omite. Esto permite interrumpir la secuencia de la línea cuando sea necesario. La inserción de puntos fuera de la pantalla es un asunto de conveniencia.

Listado 14.5a

```
7265 DRAWA   CALL  FCALL
7270         LD    HL,(PARMO)
7275         LD    (DPL+1),HL
7280 DPL     LD    A,(DPL+1)
7285         AND   128+64+32
7290         CP    128+64
7295         JR    Z,DL1
7300 ERX40   LD    DE,40
7305         CALL  ERREX
7310 DL1     LD    HL,(DPL+1)
7315         INC   HL
7320         LD    (DPLB+2),HL
7325 DPLB    LD    BC,(DPLB+2)
7330         INC   HL
7335         INC   HL
7340         LD    A,(HL)
7345         CP    2
7350         JR    Z,DL2
7355 ERX41   LD    DE,41
7360         CALL  ERREX
7365 DL2     INC   HL
7370         INC   HL
7375         INC   HL
7380         LD    A,(HL)
7385         CP    2
7390         JR    NZ,ERX42
7395         INC   HL
7400         LD    A,(HL)
7405         CP    0
7410         JR    NZ,ERX42
7415         INC   HL
7420         PUSH  HL
7425         LD    HL,-6
7430         ADD   HL,BC
7435         PUSH  HL
7440         POP   BC
7445         POP   HL
7450 NXPPR   LD    (DPLC+2),HL
7455         INC   HL
7460         INC   HL
7465         LD    (DPLD+2),HL
7470         DEC   BC
7475         DEC   BC
7480         BIT   7,B
7485         RET   NZ
7490         PUSH  BC
7495         PUSH  HL
7500 DPLC    LD    BC,(DPLC+2)
7505         LD    A,B
7510         LD    B,C
7515         LD    C,A
```

```
7520 DPLD   LD    DE,(DPLD+2)
7525        LD    A,D
7530        LD    D,E
7535        LD    E,A
7540        LD    A,E
7545        AND   128+64
7550        CP    128+64
7555        JR    Z,EXT
7560        LD    A,C
7565        AND   128+64
7570        CP    128+64
7575        JR    Z,EXT
7580        CALL  XLINE
7585 EXT    POP   HL
7590        POP   BC
7595        JR    NXPPR
7600 ERX42  LD    DE,42
7605        CALL  ERREX
```

Listado 14.5b

```
30 LET b=250
41 DIM k$(b,2)
42 FOR x=1 TO b
43 LET k$(x,1)= CHR$ 255
44 LET k$(x,2)= CHR$ 255
45 NEXT x
50 LET k=0
51 LET o1=0
53 LET l= USR movec
54 REM LEER POSICION DEL CURSOR
56 PRINT  AT 0,0;"          ": PRINT  AT 0,
0;l: POKE 23560,255
57 IF l=o1 THEN  LET l=65535
58  LET k=k+1
59  LET k$(k,2)= CHR$  INT (l/256)
60  LET k$(k,1)= CHR$  INT (l-256*( INT (l/
256)))
61 IF k=1 THEN  GO TO 58
62 LET m= USR drawa
```



```
63 REM k$();
64 LET o1=l
65 PRINT  AT 0,6;k
66 GO TO 53
```

OPERACION DE DRAWA

PCALL localiza el parámetro de la sentencia REM y se utiliza sólo el primer parámetro. Se comprueba que sea una matriz o conjunto de caracteres exactamente como está especificado. Se produce un error 40 si no es una matriz de caracteres, un error 41 si no es bidimensional y un error 42 si la segunda dimensión no es dos.

En NXPPR se obtiene el siguiente (o primer) par de puntos. HL señala al primer par de bytes. DPLC es una instrucción de carga computada, HL se adelanta dos bytes más y DPLD carga el siguiente par en DE mediante otra instrucción computada. Este será el primer par de bytes la próxima vez.

Los pares de bytes BC y DE deben ser intercambiados entre si para la llamada a XLINE. Esta inversión podría omitirse pero luego habría que invertir los pares de puntos en la tabla y esto se aparta de lo normal.

Una vez que BC y DE están ya preparados, se comprueban para asegurarse de que ambos están dentro de la pantalla. Si alguno de ellos cae fuera de la misma, se omite la rutina de trazado XLINE y se obtiene el siguiente par de puntos y se comprueba que BC sea mayor que cero.

PROGRAMA DE DIBUJO EN BASIC

Este programa, utilizando sólo MOVEC y DRAWA, permite dibujar figuras bastante complejas. Las teclas operan según lo especificado para MOVEC. La tecla «p» provoca la transferencia de la posición del cursor hacia 1 y, por tanto, hacia la parte n.° k de k$ (). Un punto repetido produce la inserción del marcador de fuera de la pantalla y el cursor puede entonces trasladarse al inicio de la línea siguiente deseada por el usuario.

Le dejo a usted con el problema de cómo interrumpir la rutina de dibujo para que pueda guardar k$ (). Sugerencia: puede reservar la parte inferior de la pantalla para el menú de alguna rutina.

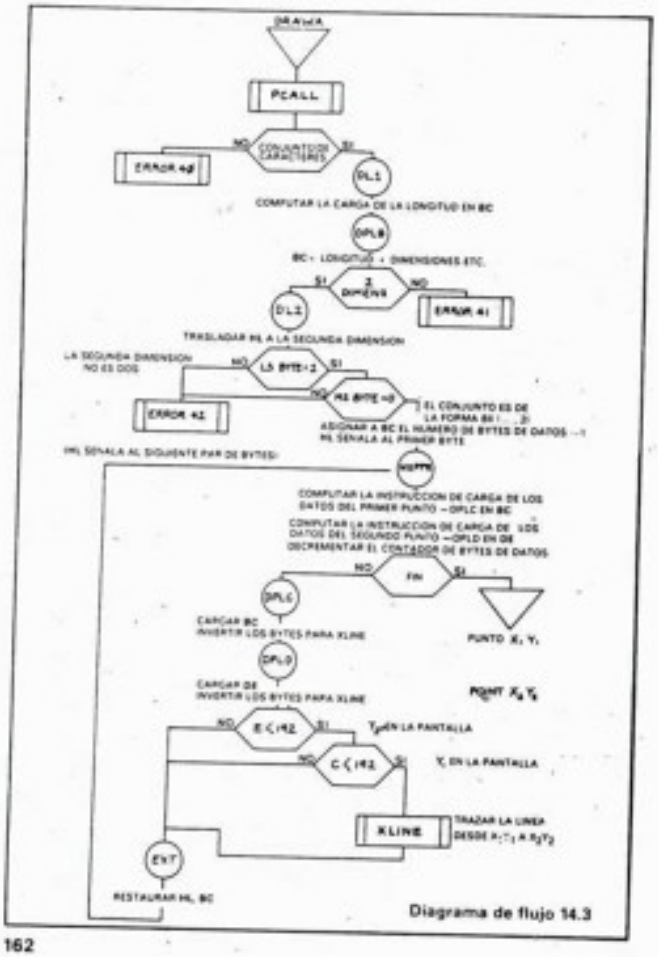


Diagrama de flujo 14.3

Sinopsis

DRAWL le permite trazar series de líneas conectadas entre sí desde el punto 1 al 2, de éste al 3, etc. Estos puntos están especificados como pares X, Y en el parámetro de la línea REM que puede ser de cualquier longitud. Por ejemplo: REM: «X1, Y1, X2, Y2, X3, Y3, X4, Y4, ... Xn, Yn,».

DRAWA es similar a DRAWL pero los datos se obtienen de un conjunto de caracteres de pares (X,Y). Los puntos que caen fuera de la pantalla no se dibujan y, por tanto, pueden interrumpirse las líneas insertando puntos fuera de la pantalla en la matriz de datos.

MOVEC utiliza las teclas 5, 6, 7, 8 para mover el cursor por la pantalla. La tecla «p» produce la salida de la rutina con la posición actual del cursor. Las teclas x, s y f permiten trabajar en el modo de paso a paso, movimiento lento y movimiento rápido del cursor.

El programa de dibujo en BASIC corresponde al *listado 14.4* y puede ser elaborado de la forma más conveniente para cada usuario.

Capitulo 15

MISCELANEA

Se presentan aqui varios detalles que son interesantes aunque no merecen un capitulo por si solos.

BCD O DECIMAL CODIFICADO BINARIO

Es una forma de aritmética y de representación numérica. Se considera de origen americano y de paternidad dudosa. Permitia a un vendedor decir a su posible victima: «... pero nuestra máquina puede trabajar con aritmética decimal. No deberia usted complicarse la vida con una de las otras. Ellas sólo pueden manejar el (molesto, complicado, dificil) binario».

Cada digito decimal puede ser representado por cuatro bits con los valores 8, 4, 2, 1 en BCD 8421. (Hay otra forma: BCD 4421.) El Z80 podrá manejar la aritmética BCD a dos digitos por byte si, después de cada operación de adición o sustracción, se inserta una instrucción DAA (Ajuste Decimal Aritmético), y se prevé una rutina especial para la escritura de los números.

Yo creo que es interesante que una máquina pueda trabajar en BCD, ya que existen muchos aparatos electrónicos que están preparados para suministrar señales codificadas en BCD —cuatro conductores por número decimal— y, por tanto, pueden interconectarse fácilmente con los sistemas de ordenadores.

MODIFICACIONES

Todo lo que he pretendido hacer en este libro es señalar al lector la dirección a seguir. No hay ningún libro que vaya a resolverle todos sus problemas pero, a modo de ilustración, incluyo algunas rutinas más cuya interpretación dejo en sus manos.

Listado 15.1

```
7610 DEMO1  CALL OPARS
7615        CALL PCALL
7620        CALL FIDL1
7625        JP   SATTR+3
7630 FIDL1  LD   HL,(PARMO)
7635        LD   BC,3
7640        ADD  HL,BC
7645        LD   DE,VPARO+8
7650        LDI
7655        RET
7660 DEMO2  CALL PCALL
7665        CALL FIDL2
7670        JP   SATTR+3
7675 FIDL2  LD   HL,(PARMO)
7680        LD   BC,8
7685        ADD  HL,BC
7690        LD   DE,VPARO
7695        CALL LDPR
7700        CALL LDPR
7705        CALL LDPR
7710        CALL LDPR
7715        CALL LDPR
7720        RET
7725 LDPR   LDI
7730        LD   BC,4
7735        ADD  HL,BC
7740        INC  DE
7745        RET
7750        END
```

DEMO1

Esta rutina entra en SATTR después de la llamada a OPARS y PCALL. La sentencia REM que espera encontrar es:

REM k : 0, 0, 15, 7

donde k es un atributo (entero) y las constantes son para definir la región de la pantalla.

DEMO2

Esta rutina también entra en SATTR pero la sentencia REM es:

REM a() :

y los primeros cinco elementos de a() son los que describen la región de la pantalla y el atributo requerido. Deben ser todos enteros. Ambas rutinas utilizan subrutinas auxiliares. Observe su sencillez, procure entender la forma en que actúan y disfrute poniendo algo de su parte.

ENTRADAS MULTIPLES

Con una gran cantidad de programas, pueden ocurrir cosas desagradables:

— el programa A sale por la pantalla 1
— el programa B sale por la pantalla 2

Hay una subrutina común, C, en las profundidades del programa, que se encarga de la gestión de la pantalla.

A está dando salida a la pantalla cuando ésta tiene algún defecto y lo comunica a C, la cual emite un mensaje de error para el operador y espera a que se corrija el defecto en la pantalla.

El programa-B da ahora su salida a la pantalla utilizando la subrutina C y pronto se encuentra en dificultades a menos que C haya sido prevista para encargarse del problema.

La técnica utilizada con más frecuencia consiste en estimar primeramente el número de llamadas múltiples que pueden actuar al mismo tiempo, sumarle un 50 % (o más) y preparar después este número de páginas, utilizando el registro IX o equivalente, para todo el espacio de trabajo necesario para una entrada. Cada celda se encarga de una «página» que deja luego cuando la llamada ha terminado. Si no hay espacio disponible, el programa que está llamando debe ser informado de ello para que espere o haga cualquier otra cosa hasta que su llamada pueda ser aceptada.

LA ACCION RECURSIVA O LA SERPIENTE QUE SE MUERDE LA COLA

La acción recursiva es la que se produce cuando una subrutina se llama a sí misma. Esto puede ocurrir por accidente en programas muy

extensos o de una forma deliberada como resultado de un intento de reducción del número de instrucciones u otro caso similar. Siempre requiere una gran cantidad de espacio en la pila.

Normalmente, una llamada de una subrutina a si misma destruirá los espacios de trabajo y la dirección de retorno. Por tanto, la rutina debe estar preparada para superar este contratiempo. En algunos casos, el problema es similar al de las entradas múltiples pero aquí los datos están todos almacenados en la pila y una sección de ésta se utiliza también como espacio de trabajo. La técnica básica se ilustra en la *Figura 15.1*. Usted debe asegurarse de que la llamada de la rutina a si misma sea condicional y que, si la condición no se cumple, la subrutina podrá abandonar el círculo vicioso y salir al mundo exterior. Si no se consigue esto, como la serpiente que se muerde la cola, tendrá también un final desagradable.

NOTAS SOBRE EL CODIGO MAQUINA Y EL ENSAMBLADOR

Todos los mnemónicos de los códigos de operación son estándar. Las operaciones «escondidas», es decir, aquellas con las cuales opera el hardware pero cuya existencia no es oficial, también se utilizan.

Los mnemónicos utilizados para controlar el ensamblador son:

DEFB	define un byte como un número decimal o un carácter ASCII
DEFS	define una serie de bytes utilizando una cadena ASCII
DEFW	define una palabra de dos bytes
END	especifica el final del código máquina
EQU	requiere una etiqueta, la cual es asignada a una dirección, que es normalmente la de una variable del sistema del Spectrum
ORG	especifica la dirección de inicio del código ensamblado.

Un valor de un solo byte puede ser especificado como un valor decimal (0-255) o un carácter ASCII encerrado entre comillas. Nótese que LD A, """" cargará A con el código ASCII de " ".

CODIGO MAQUINA — LO QUE DEBE Y NO DEBE HACER

Ensamble el código para que funcione en zonas altas de la memoria pero deje suficiente espacio para la pila entre el final del código

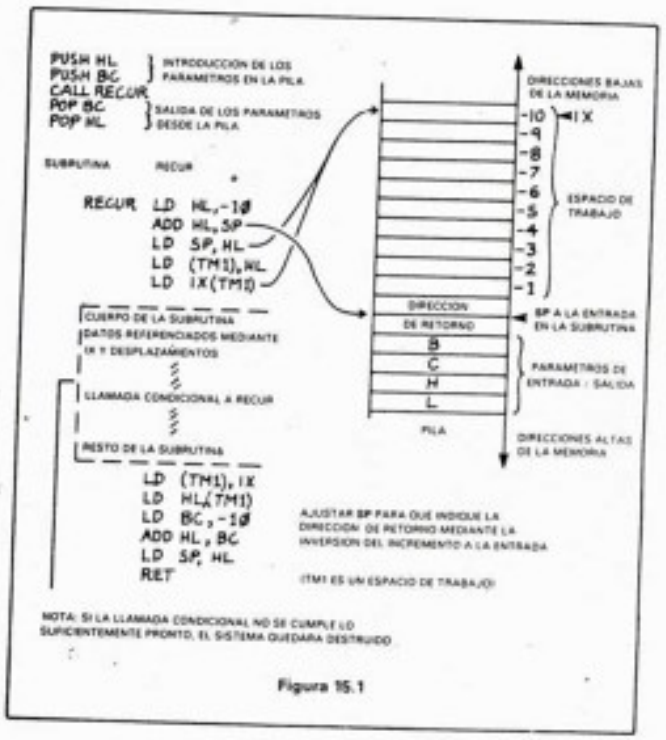


Figura 15.1

y la zona de Gráficos Definidos por el Usuario (UDG) del Spectrum (Ver el Capítulo 24 del manual). En general, todo irá bien si el final de su código se encuentra alrededor de la posición 63500 en una máquina de 48 K. No utilice *nunca* direcciones absolutas (valores numéricos) en su código. Las direcciones absolutas deberían utilizarse solamente para referirse a las variables del sistema del Spectrum (detalladas en el Capítulo 25 del manual) o bien a partes específicas de la ROM.

¡Guarde anotaciones en sus programas, especialmente las que se refieran a sus errores!

Siempre que sea posible conviértanse todos los nombres en mnemónicos.

Escriba sus programas lo más directa y claramente posible. (Un programa que funciona es mejor que nada. (¡Hay pocos conductores que miren debajo del capó!)

Tenga siempre una idea clara de lo que desea hacer antes de empezar.