CONTROL DE

Sin duda, uno de los factores más decisivos para que un programa tenga éxito es su presentación. Si la idea es muy original, pero los gráficos son tamaño UDG o parpadean, el conjunto desmerece bastante. Para solucionar el problema presentamos una rutina que moverá y dibujará los gráficos mientras nuestro programa se encarga de otras tareas.

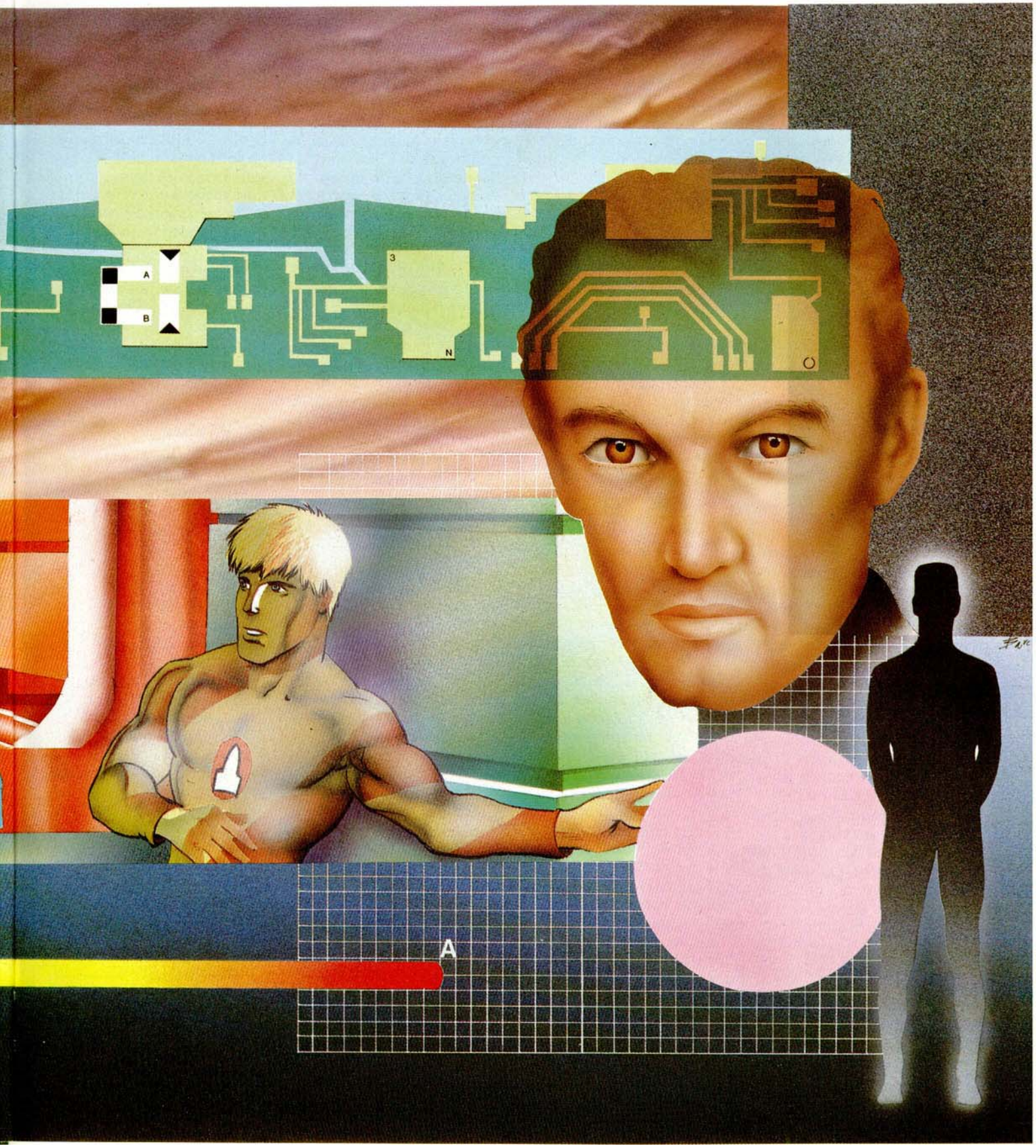
En primer lugar, para los no iniciados, definiremos qué es lo que entendemos por un sprite.

Llamaremos sprite a un gráfico, animado o no, que se mueve por la pantalla en una dirección determinada. Este gráfico deberá pasar por delante de lo que haya dibujado en pantalla sin borrarlo, y si se cruza con otro, uno de los dos pasará por delante de otro. Por esto cada sprite tiene asignada una prioridad, de tal forma que al cruzarse dos sprites, pasará por delante aquél cuya prioridad sea mayor. Evidentemente, no puede haber dos sprites con la misma prioridad.

En otros ordenadores, como el Commodore 64, los sprites se generan por hardware, con lo cual no se consume tiempo de programa. Pero lamentablemente, en el Spectrum no existe hardware para la creación de sprites, así que no tendremos más remedio que crearlos por software. Para ello, utilizaremos un programa en

SPRITES

Pablo ARIZA



Código Máquina que se ejecutará automáticamente gracias a las interrupciones. Sobre interrupciones ya se ha hablado bastante en números anteriores de esta revista, así que bastará con que digamos que sirven para que una subrutina escrita en Código Máquina se ejecute automáticamente 50 veces por segundo. Cuando termina, se devuelve el control al programa que estaba ejecutándose antes de producirse el salto a la subrutina. De esta forma, podemos lograr el efecto de que se están efectuando dos tareas a la vez. En nuestro caso, una de las tareas será el dibujo de los sprites, y la otra cualquier programa escrito por nosotros, ya sea en Basic o Código Máquina.

Normalmente, las subrutinas que se ejecutan por interrupciones suelen ser muy cortas, de tal forma que aparentemente la velocidad del programa principal es la misma que tendría si no hubiera interrupciones. Sin embargo, el dibujo de gráficos y el manejo de la pantalla en general, es siempre bastante lento, y cuando pongamos en marcha los sprites, notaremos un descenso enorme en la velocidad de ejecución de nuestros programas. Es el precio que hay que pagar por no tener el hardware adecuado. A pesar de todo, si programamos en Código Máquina, tendremos suficiente velocidad para bastantes cosas, pues no hay que olvidar que nuestro programa ya no se tendrá que preocupar de dibujar los gráficos, que suele ser el proceso que más tiempo consume en la ejecución de un programa, sobre todo en los juegos arcade.

El método que vamos a utilizar para dibujar los gráficos es el de la máscara. En este método, cada gráfico se compone en realidad de dos gráficos: el gráfico propiamente dicho y su máscara. La máscara es un gráfico adicional, de las mismas dimensiones que el primero, y que nos facilita información acerca de la forma de éste. Esto sirve para que cuando el sprite pase por delante de algo, no se mezcle su imagen con la del fondo ni se borre un rectángulo del fondo alrededor del sprite, como suele ocurrir en los métodos tradicionales de dibujo de sprites. Es el método que se suele usar en los juegos tridimensionales, como «Knight Lore», y en algunos otros, como «Everyone's Wally». Tiene el inconveniente de ocupar más memoria (cada gráfico ocupa el doble, pues necesita de su máscara) y de ser más lento el proceso de dibujo, pero los resultados obtenidos son mucho más espectaculares. Para los que leyeron los artículos sobre el sistema Filmation, publicados en los números 96,97,99 y 100 de MICROHOBBY, en los que se utilizaban también máscaras, advertimos que en esta rutina vamos a utilizar un método ligeramente distinto. Para evitar el paso de inversión de la máscara descrito en dichos artículos, nosotros almacenaremos la máscara ya invertida, con lo que ahorramos tiempo a la hora de dibujar el gráfico. Por tanto, si teneis hechos de antemano gráficos pensados para la rutina presentada en dichos artículos y los queréis usar con ésta, deberéis invertir todos los bytes de la máscara. La forma de hacerlo es bien sencilla. Si por ejemplo, tenéis una máscara en la dirección 40000 que ocupa 32 bytes, bastará con que hagáis:

```
FOR X=40000 TO 40031
:POKE X,255-PEEK X:
NEXT X
```

Si vais a crear gráficos nuevos, específicamente para esta rutina, la forma de crear la máscara, una vez realizado el gráfico, podría ser así:

Observamos la figura 1. En primer lugar, alrededor del gráfico terminado (parte A), dibujamos una línea que marque su contornos, obteniendo la parte B. A continuación, rellenamos desde aquí hacia afuera todo el rectángulo que contiene el gráfico, obteniendo la parte C. Por último, borramos el gráfico original (naturalmente, antes lo habremos grabado), y nos queda la parte D, que es ya la máscara. Si el gráfico tuviera agujeros internos (por ejemplo, si dibujamos una rosquilla), deberemos hacer el mismo proceso con los agujeros. Lo que representa la máscara son las zonas del gráfico a través de las que se puede ver el fondo.

TABLA DE SPRITES

Para conseguir que nuestro sprite se mueva como nosotros queremos, habremos de comunicarle de algún modo a la rutina de control todos los datos necesarios acerca de él. Con este fin, vamos a crear una tabla de sprites, en la que iremos poniendo la posición, dimensión, y demás datos de cada uno de ellos. La rutina está pensada para manejar un máximo de 17 sprites, que numeraremos del 0 al 16 (este número servirá asimismo para indicar la prioridad del sprite). Como veremos enseguida, para especificar todos los datos de un sprite necesitaremos 17 bytes, o lo que es lo mismo, un total de $17 \times 17 = 289$ bytes, que los colocaremos en la dirección 53000. De esta forma, los datos del primer sprite estarán a partir de la dirección 53000, los del segundo en la 53017, etc.

Vamos a ver ahora qué es lo que deberemos meter en cada uno de esos grupos de 17 bytes. Tenemos un resumen en la figura 2. Los dos primeros servirán para indicar la dirección de memoria en la que se encuentra el gráfico. Siempre, a continuación del gráfico propiamente dicho, se debe encontrar su máscara. Si el sprite representa un objeto o un ser animado, deberá constar de varios gráficos parecidos, pero distintos para lograr el efecto de animación. En este caso, todos estos gráficos deben estar uno a continuación del otro, y cada uno con su máscara detrás.

El tercer y cuarto byte indican las dimensiones del sprite. El tercero indicará el ancho, dado en caracteres, y el cuarto el alto, dado en píxels. Todos los gráficos de

FIGURA 1

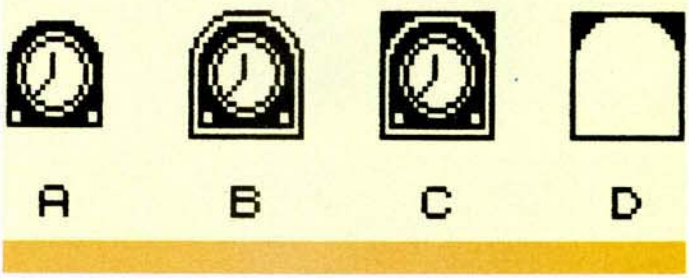


Fig. 2

TABLA DE DATOS DE LOS SPRITES

Dirección relativa	Contenido
DIR+Ø	Dirección del primer gráfico.
DIR+2	Ancho en caracteres.
DIR+3	Alto en scans (pixels).
DIR+4	Coordenada X.
DIR+6	Coordenada Y.
DIR+8	Número de fases de animación.
DIR+9	Color. Bit 3:Animación. Bit 4:Movimiento.
DIR+1Ø	Incremento X. DIR. Tabla de trayectoria.
DIR+11	Incremento Y.
DIR+12	Contador de animación.
DIR+13	Contador de trayectoria.
DIR+15	Memoria ocupada por cada gráfico.

las distintas fases de un único sprite y sus respectivas máscaras deberán tener las mismas dimensiones.

Los bytes quinto y sexto especifican la coordenada X del sprite, mientras que los séptimo y octavo especifican la Y. Ambas serán números entre —32768 y +32767. Esto nos permite crear un sprite inicialmente en unas coordenadas que no existan en la pantalla real del ordenador, y hacer que aparezca posteriormente. Si la coordenada X está entre 32512 y 32767 (o lo que es lo mismo, si el byte sexto es 127), se considerará al sprite como desactivado y ni se le moverá ni se le dibujará. A diferencia del Basic, la coordeanda Y comienza por Ø en la parte superior y va aumentando hacia abajo.

El noveno byte nos dice cuántas fases de animación tiene el sprite. Si se trata de un sprite inanimado, pondremos una fase.

El décimo indica varias cosas. En primer lugar, indica el color del sprite. Los tres primeros bits indican el color de la tinta, y el séptimo bit (el bit 6) indica el nivel de brillo. Los sprites no tienen color de papel propio; adoptan el del fondo por el que pasan. Por otra parte, el cuarto bit (el bit 3) indica, en el caso de sprites animados, de qué tipo de animación se trata. Para sprites inanimados, su valor es indiferente. Si vale Ø la animación será cíclica, mientras que si vale 1, será una animación de adelante-atrás. La primera es la que usaríamos, por ejemplo, para un helicóptero cuyas hélices giran. La segunda la usaríamos, por ejemplo, para un pez que mueve la cola. Si tenemos cinco fases de animación, que podemos numerar del 1 al 5, y especificamos animación cíclica, las fases se irán repitiendo de esta forma:

1 2 3 4 5 1 2 3 4 5 1 2 3 4 5...

Si, por el contrario, especificamos animación de adelante-atrás, obtendremos la secuencia:

1 2 3 4 5 4 3 2 1 2 3 4 5 4 3 2 1 2...

Por último, en este décimo byte, está también, en el bit 4 el indicador de línea recta (con un Ø) y trayectoria compleja (con un 1). La opción de línea recta se usará, como su propio nombre indica, para sprites que se muevan en línea recta, sea horizontal, vertical o diagonal. Sin embargo, habrá casos en que preferiremos que nuestro sprite siga una trayectoria preestablecida distinta de la línea recta, por ejemplo, un satélite que gira en círculo alrededor de un planeta. Para esto sirve la opción de trayectoria. La forma de establecer la trayectoria la vamos a ver a continuación.

Fig. 3

Gráficos usados en la demostración

Dirección	Gráfico	Dimensiones (en caracteres)	Número de fases
47300	COHETE	2×2	1
47364	RELOJ	3×3	8
48516	FLECHA →	2×2	1
48580	FLECHA ↗	2×2	1
48644	FLECHA ↑	2×2	1
48708	FLECHA ↖	2×2	1
48772	FLECHA ←	2×2	1
48836	FLECHA ↙	2×2	1
48900	FLECHA ↓	2×2	1
48964	FLECHA ↘	2×2	1

El significado de los bytes undécimo y decimosegundo no es el mismo si el sprite se mueve en línea recta o siguiendo una trayectoria especial.

En el caso de un movimiento en línea recta, estos bytes indican los incrementos de X e Y respectivamente. Dichos incrementos son números entre —128 y +127, que se suman cada vez a las coordenadas del sprite para producir el movimiento. Cuanto mayor sea su valor absoluto, mayor será la velocidad con que se mueva el sprite, pero menor su suavidad. Los incrementos más normales están entre —4 y +4. En la coordenada X, un incremento positivo producirá un movimiento hacia la derecha, y negativo, hacia la izquierda. En la Y, un incremento positivo producirá un movimiento hacia abajo, y negativo hacia arriba.

En el caso de un movimiento de trayectoria prefijada, estos dos bytes indican la dirección donde se encuentran los datos acerca de la misma. En esta dirección pondremos dichos datos de la siguiente forma: para delimitar la trayectoria, se colocan grupos de dos bytes que indican los incrementos X e Y a efectuar en cada momento. Pero además, puede que queramos que el sprite cambie su forma durante el movimiento. Por ejemplo, si queremos dibujar una nave girando, no bastará con dar los incrementos que delimiten la trayectoria de giro, sino que, además, deberemos ir dibujando un gráfico un poco girado respecto del anterior para que parezca que la nave está realmente girando. Para ello, entre los grupos de dos bytes que delimitan el movimiento, colocaremos un 126, y a continuación, la dirección de memoria donde se encuentra el nuevo gráfico. Para terminar la trayectoria, colocaremos un 127 y ésta se repetirá desde el principio.

El byte decimotercero es utilizado como contador pa-

ra la animación, y deberemos inicializarlo a Ø.

Los bytes decimocuarto y decimoquinto se usan como contador para la trayectoria. Si estamos utilizando un movimiento en trayectoria preestablecida, deberemos inicializarlos con los mismos valores que los bytes undécimo y decimosegundo, es decir, con la dirección de los datos de la trayectoria. Si el movimiento es rectilíneo, estos bytes no se usan.

Por último, los bytes decimosexto y decimoséptimo indican la cantidad de memoria que ocupa cada gráfico del sprite, ya sea gráfico propiamente dicho o máscara, y se calcula multiplicando el ancho en caracteres por el alto de píxels. Este valor es para el uso interno de la rutina, y no habría sido necesario incluirlo en la tabla de sprites, pues se puede calcular a partir de los contenidos de los bytes tercero y cuarto, pero se ha incluido por razones de velocidad.

EL PROGRAMA

Antes de profundizar más en el funcionamiento de la rutina, sería preferible teclear ahora el programa para ver los resultados y comprender así mejor cómo funciona. En primer lugar, habremos de teclear el listado 1 y grabarlo en cinta con autoejecución en la línea 9999. A continuación, y con ayuda del Cargador Universal de Código Máquina, publicado innumerables veces en Microhobby y Micromanía, teclearemos el listado 2 y lo grabaremos en cinta, a continuación, del Basic con el nombre «Sprite.Code Data», indicando como dirección de comienzo 47140, y como longitud 1888. Ahora podemos teclear el listado 3 en el Cargador Universal de Código Máquina, o el listado 4 con un ensamblador, preferiblemente el Gens, y ensamblar. En ambos casos, grabaremos el resultado a continuación de lo anterior como «Sprite.Code USR», indicando como dirección la 61953, y como longitud 1495. Ahora, ya lo tenemos todo listo. Rebobinemos y carguemoslo todo. El verdadero programa controlador de sprites es el del listado 3 ó el 4. El Basic y el listado 2 sirven para efectuar una demostración de sus posibilidades, que son las que veremos al finalizar la carga. Dicha demostración se compone de siete ejemplos, desde el más simple hasta el más complejo. Estos ejemplos no están pensados solamente para que veamos cómo se mueven los sprites, sino para que, estudiando el programa Basic, podamos comprender mejor cómo manejarlos desde nuestros propios programas. En él podemos ver cuál es la secuencia a realizar para que los sprites comiencen a moverse: en primer lugar, se introducen los datos de cada uno de los sprites que queramos crear en la tabla anteriormente explicada. Es necesario advertir que si queremos, por ejemplo, crear tres sprites, estos deberán ser los números Ø, 1 y 2, es decir, los tres primeros. El programa Basic utiliza una subrutina que lee los datos de líneas DATA y hace los POKEs pertinentes, pero cada uno puede hacerse su propia subrutina como quiera, teniendo en cuenta cuáles son los datos que hay que indicar, y que ya han sido explicados. Una vez creados los datos de los sprites, activaremos las interrupciones con RANDOMIZE USR 61953 e indicaremos el programa controlador de sprites, cuántos sprites estamos utilizando, haciendo POKE 23681, N, donde N es el número de sprites. A partir de ese momento, los sprites comenzarán a moverse.

Para evitar parpadeos y otras fealdades varias, todos los dibujos se hacen en una copia de la pantalla que se encuentra en otra dirección de memoria, y luego el resultado se vuelca a la pantalla del ordenador. Esta copia de la pantalla es inicializada al activarse las interrupciones, así que si queremos que nuestros sprites se muevan sobre algún fondo o dibujo, tendremos que dibujarlo en la pantalla, justo antes de hacer el RANDOMIZE USR 61953. Una consecuencia del uso de una pantalla en memoria, es que no podremos dibujar ni escribir nada en la pantalla del ordenador, porque las interrupciones se encargan de copiar constantemente la pantalla de memoria en la pantalla del ordenador, así que nada más escribir algo, se borrará automáticamente. Para poder escribir algo, deberemos hacerlo antes de activar las interrupciones, o escribiendo directamente en la pantalla de memoria, para lo cual veremos más adelante dónde y cómo está colocada.

Independientemente de la rutina de sprites, hay una subrutina que puede resultar muy útil en algunos casos. Se trata de la que comprueba el choque entre dos sprites. La forma de utilizarla es la siguiente: POKE 23729,A:POKE 23728,B:

LET C=USR 62082, donde A y B son los códigos de dos sprites, y C terminará valiendo 1 si los dos sprites están chocando o 0 si no lo están. Si lo que queremos es averiguar si un sprite está chocando con cualquier otro, sin importarnos cuál, sustituiremos B por un 255.

Comentemos un poco los ejemplos de la demostración, porque ilustran bastante bien las características de los sprites.

El primer ejemplo es el más sencillo. Un único sprite de color rojo y sin animación aparece a la izquierda de la pantalla y se mueve hacia la derecha.

El segundo ejemplo nos muestra una especie de reloj con una aguja siempre quieta y la otra girando en sentido horario. Como deberiáis haber supuesto, se trata de animación cíclica. Además, podéis comprobar cómo un sprite no necesita estar siempre moviéndose, puede estar en una posición fija, poniendo los incrementos X e Y a 0. En este modo, podemos controlar directamente la posición del sprite, pokeando en los lugares oportunos, en este caso, las direcciones 53004 Y 53006.

En el tercer ejemplo, vemos como actúa la máscara para producir un efecto muy convincente de que el sprite está por delante del fondo. Desgraciadamente, podemos comprobar también cómo en el Spectrum es imposible evitar la famosa mezcla de colores, pero ésta es reducida al mínimo posible.

El cuarto ejemplo consta ya de dos sprites para ver cómo al cruzarse uno pasa por delante del otro. Podemos comprobar también cómo el sprite toma el color de papel que haya en la pantalla, lo que contribuye a evitar un poco la mezcla de colores que acabamos de mencionar.

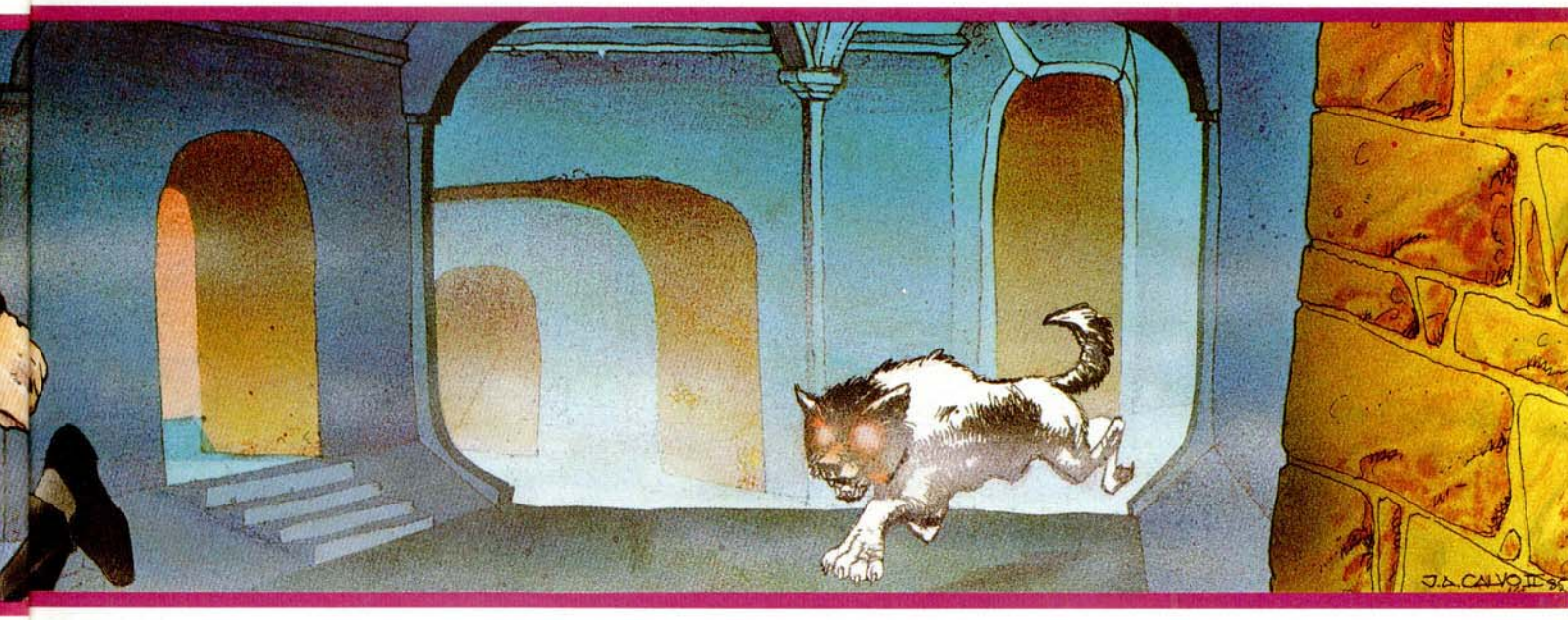
El quinto ejemplo muestra, por una parte, un ejemplo práctico del uso de trayectorias preestablecidas con una flecha que gira, y por otra parte un grupo de cuatro flechas paradas de las que dos tienen animación cíclica y dos de adelante-atrás, para que se puedan apreciar bien las diferencias entre una y otra.

El sexto ejemplo muestra lo llena que puede parecer la pantalla cuando se colocan los 17 sprites circulando en todas direcciones.

Por último, el séptimo no se limita a crear los sprites y dejar que se muevan, sino que, con la ayuda de la subrutina y de comprobación de choque, produce unos sonidos cada vez que el cohete es interceptado por una flecha. Es un ejemplo muy simplificado de lo fácil que es hacer un juego utilizando esta rutina para controlar los sprites.

Para hacer vuestras primeras pruebas, podéis utilizar también el programa Basic de demostración, con sólo cambiar los DATAs de algún ejemplo. Para ayudaros, tenéis en la figura 3 una tabla con las direcciones y demás datos de los gráficos usados para la demostración, y en la figura 4 un resumen de qué valores y en qué orden debéis poner en los DATAs. Advertencia importante: al final de los datos de todos los ejemplos, es necesario añadir un cero a los DATAs para que el programa distinga entre los datos de un ejemplo y los de otro, y no los lea todos de una vez.

La rutina de interrupciones, la pantalla de memoria, y algunas zonas de datos más, ocupan por completo de la dirección 53000 en adelante, así que nuestros programas y gráficos deben situarse por debajo de aquí. Puede parecer que ocupa demasiado, pero, de hecho, es más o menos lo que suelen ocupar las rutinas y áreas de datos destinados al mismo fin en programas comerciales. Hay que pensar que solamente la pantalla de memoria ocupa ya unos 7 K. Pero si todavía os parece mucho, vamos a ver con detenimiento qué es lo que hay a partir de esa dirección.



MAPA DE MEMORIA

Si sólo pensáis usar la rutina desde Basic y no tenéis ningún interés en el Código Máquina, no es necesario que leais este apartado ni el siguiente, pues en innecesario lo que se dice en ellos

para poder utilizar los sprites. Pero si sabéis Código Máquina, continuad leyendo y podréis sacar mucho más provecho a la rutina.

En la figura 5 tenéis un esquema del mapa de memoria. Lo primero que tenemos, entre las direcciones 53000 y 53288, es la tabla de sprites, ya explicada anteriormente, y cuyo inicio será denominado TASPRI a partir de ahora.

Lo siguiente, entre 53288 y 53543, es otra tabla, TABLDI, que no hemos explicado antes porque es de uso interno de la rutina. En esta se almacenan una serie de datos de los sprites, pero solamente de los que se van a dibujar en pantalla (como hemos visto, puede haber sprites fuera de ésta). En la figura 6 hay un resumen de los 15 bytes de que consta cada elemento. Los dos primeros bytes indican la dirección en la pantalla de memoria a la que va a ir el gráfico. Los dos siguientes indican la dirección del gráfico, y los otros dos, la de la máscara. El séptimo byte indica el numero de scans, o lo que es lo mismo, la altura en píxels. El octavo byte indica el ancho en caracteres. Tanto éste como el anterior no tienen por qué corresponder con los dados en la tabla de sprites, ya que éstos se refieren a las dimensiones del gráfico que se va a dibujar en pantalla, y si éste se encuentra en un lado o una esquina, la porción de gráfico que se ve es menor que el gráfico entero. El noveno byte nos dice cuántos caracteres de ancho quedan fuera de la pantalla. Comprobaremos su utilidad al estudiar el programa. El décimo byte es el llamado byte de rotación. Puede valer 248, 250, 252 ó 254, y es el byte alto de la dirección de comienzo de una tabla. Son, por tanto, cuatro tablas. La primera va de 63488 a 63999, la segunda de 64000 a 64511, la tercera de 64512 a 65023, y la cuarta de 65024 a 65535. Las cuatro se crean en el momento de activar las interrupciones. Lo que hay en ellas es el resultado de girar cada uno de los 256 posibles números que puede contener un byte, 6 bits hacia la derecha, en la tabla cuarta, 4 en la tercera, 2 en la segunda y 0 en la primera. La utilidad es que a la hora de dibujar un gráfico, no tendremos que utilizar instrucciones de rotación que hacen más lento el proceso, puesto que ya tenemos el trabajo hecho. Por ejemplo, si queremos girar el número 47, 4 bits a la derecha (el número 47 puede formar parte de un gráfico, ya que sabemos que todos los gráficos, en el ordenador se reducen a números binarios, con un equivalente decimal), el proceso a efectuar sería cargar el registro H con 252, que es el número que corresponde a la tabla tercera, después cargar l con 47, que es el número que queremos girar, y la dirección formada contendrá el resultado deseado. Como al girar un byte, hay una parte que se sale, ésta se almacena en otra dirección, justo 256 bytes más arriba, así que basta con incrementar H para obtenerla. Si no os ha quedado demasiado claro, posiblemente lo entendais mejor cuando veais la subrutina que crea la tabla. Como muchos de vosotros ya habréis adivinado, el que sólo haya cuatro tablas quiere decir que el programa no trabaja realmente en alta resolución en este sentido horizontal, sino que el mínimo desplazamiento es de dos píxels. Esto no supone un gran problema, pues la suavidad obtenida es más que suficiente. Sin embargo, para mayor comodidad se permite que se den las coordenadas como si el mínimo desplazamiento fuera de dos píxels. La verdad es que este sistema del uso de tablas con los bytes desplazados no es de nueva creación, ha sido utilizado en varios programas comerciales, entre los que cabe destacar «Cyberun», cuyo rápido scroll habría sido imposible con métodos convencionales. Pero continúemos con el contenido de la tabla TADIBL. El undécimo byte indica los atributos del sprite. El decimosegundo y el decimotercero indican la dirección en la pantalla de memoria, donde van a ir los atributos del sprite. Por último, los bytes decimocuarto y decimoquinto indican el número de filas y el de columnas, respectivamente, que ocupa el gráfico en la zona de atributos.

Continuando con el mapa de memoria, entre las direcciones 53544 y 54566, está el espacio denominado

Fig. 4

Valores a colocar en los DATAS en el programa Basic

- Dirección del primer gráfico.
- Ancho en caracteres.
- Alto en pixels.
- Coordenada X.
- Coordenada Y.
- Número de fases de animación.
- Indicador: 0 = animación cíclica, 1 = adelante/atrás.
- Indicador: 0 = línea recta, 1 = trayectoria compleja.
- Color (tinta + 64 × brillo).
- Si línea recta:
  - Incremento X.
  - Incremento Y.
- Si trayectoria compleja:
  - Dirección de los datos de la trayectoria.

Fig. 5

Mapa de memoria

Dirección	Etiqueta	Contenido
53000	TASPRI	Tabla de datos de los sprites existentes.
53289	TABLDI	Tabla de datos de los gráficos que se van a dibujar.
53544	SPARES	Espacio reservado para guardar los trozos de pantalla.
54568(7)	ORIGSC	Pantalla de memoria.
60904	ORIGAT	Atributos de la pantalla de memoria.
61696		Tabla de interrupciones.
61953	INICIO	Subrutina de inicialización y activación.
62075	DESACT	Subrutina de desactivación.
62082	COMCHO	Subrutina de comprobación de choque.
62194	ENTINT	Subrutina principal de interrupciones.
63488		Tablas de rotaciones.

LISTADO 1

```
  1 GO TO 1000
 10 RANDOMIZE USR 62075: CLS :
LET NUMSPR=23681: LET N=0: LET A
=53000
 20 READ D: IF D=0 THEN RETURN
 30 RANDOMIZE D: POKE A,PEEK 23
670: POKE A+1,PEEK 23671
 40 READ H,V: POKE A+2,H: POKE
A+3,V: RANDOMIZE H*V: POKE A+15,
PEEK 23670: POKE A+16,PEEK 23671
 50 READ X,Y: POKE A+5,INT (X/2
56): POKE A+4,X-256*INT (X/256):
 POKE A+7,INT (Y/256): POKE A+6,
Y-256*INT (Y/256)
 60 READ M,C: POKE A+8,M: POKE
A+12,0
 70 READ R,CO: POKE A+9,CO+C*8+
R*16: IF R=1 THEN READ D2: RANDO
MIZE D2: POKE A+10,PEEK 23670: P
OKE A+11,PEEK 23671: POKE A+13,P
EEK 23670: POKE A+14,PEEK 23671:
 GO TO 90
 80 READ IX,IY: POKE A+10,IX: P
OKE A+11,IY
 90 LET N=N+1: LET A=A+17: GO T
O 20
 100 REM EJEMPLO 1
 110 DATA 47300,2,16,0,88,1,0,0,
2,2,0,0
 120 REM EJEMPLO 2
 130 DATA 47364,3,24,120,80,8,0,
0,1,0,0,0
 140 REM EJEMPLO 3
 150 DATA 48580,2,16,-32,192,1,0
,0,2,2,-2,0
 160 REM EJEMPLO 4
 170 DATA 47300,2,16,0,88,1,0,0,
1,3,0
 180 DATA 47364,3,24,256,80,8,0,
0,0,-2,0,0
 190 REM EJEMPLO 5
 200 DATA 48516,2,16,100,100,1,0
,1,2,47140
 210 DATA 48516,2,16,0,0,8,0,0,0
,0,0
 220 DATA 48516,2,16,16,16,8,0,0
,0,0,0
 230 DATA 48516,2,16,0,16,8,1,0,
0,0,0
 240 DATA 48516,2,16,16,0,8,1,0,
0,0,0,0
 250 REM EJEMPLO 6
 260 DATA 48516,2,16,0,0,8,0,0,0
,2,1
 270 DATA 48516,2,16,0,16,8,1,0,
0,2,1
 280 DATA 47300,2,16,0,80,1,0,0,
4,3,0
 290 DATA 48516,2,16,-66,20,8,1,
0,0,1,0
 300 DATA 48516,2,16,-50,36,8,1,
0,0,1,0
 310 DATA 47300,2,16,0,0,1,0,0,0
,6,0
 320 DATA 48516,2,16,-66,36,8,0,
0,0,1,0
 330 DATA 48516,2,16,-50,20,8,0,
0,0,1,0
 340 DATA 47364,3,24,0,255,8,0,0
,0,0,-4
 350 DATA 48580,2,16,0,200,1,0,0
,2,2,-2
 360 DATA 48708,2,16,255,255,1,0
,0,3,-3,-3
 370 DATA 47364,3,24,255,72,8,0,
0,5,-2,0
 380 DATA 48516,2,16,90,100,1,0,
1,0,47140
 390 DATA 48516,2,16,0,20,1,0,1,
2,47140
 400 DATA 47364,3,24,128,88,8,1,
0,0,0,0
 410 DATA 47300,2,16,-60,255,1,0
,0,1,1,-1
 420 DATA 48516,2,16,90,124,1,0,
1,2,47140,0
 430 REM EJEMPLO 7
 440 DATA 47300,2,16,0,160,1,0,0
,3,2,0
 450 DATA 48900,2,16,0,0,1,0,0,0
,0,2
 460 DATA 48900,2,16,32,0,1,0,0,
0,0,5
 470 DATA 48900,2,16,64,-24,1,0,
0,0,0,3
 480 DATA 48900,2,16,96,16,1,0,0
,0,0,3
 490 DATA 48900,2,16,128,24,1,0,
0,0,0,3
 500 DATA 48900,2,16,160,0,1,0,0
,0,0,2
 510 DATA 48900,2,16,192,-32,1,0
,0,0,0,2
 520 DATA 48900,2,16,224,-164,1,
0,0,0,0,3,0
1000 PRINT TAB 3;"SPRITES POR IN
TERRUPCIONES";;TAB 4;"PROGRAMA
 DE DEMOSTRACION";#0;" PULSA UNA
 TECLA PARA CONTINUAR"
1010 PAUSE 0
1020 RESTORE : GO SUB 10: LET X=
1: GO SUB 1120: PAUSE 256
1030 GO SUB 10: LET X=2: GO SUB
1120: PAUSE 200
1040 GO SUB 10: FOR X=0 TO 255 S
TEP 4: PLOT 0,0: DRAW X,175: PLO
T 255,175: DRAW -X,-175: NEXT X:
LET X=3: GO SUB 1120: PAUSE 256
1050 GO SUB 10: FOR X=0 TO 21: F
OR Y=3 TO 6: PRINT PAPER Y;"
";: NEXT Y: NEXT X: LET X=4:
GO SUB 1120: PAUSE 290
1060 GO SUB 10: LET X=5: GO SUB
1120: PAUSE 200
1070 GO SUB 10: PRINT : FOR X=1
TO 21*32: PRINT CHR$ (65+INT (RN
D*28));: NEXT X: LET X=6: GO SUB
 1120: PAUSE 600
1080 GO SUB 10: LET X=7: GO SUB
1120: POKE 23729,0: FOR X=1 TO 9
0
1090 POKE 23728,255: IF USR 6208
2 THEN BEEP .01,RND*60
1100 NEXT X
1110 RANDOMIZE USR 62075: PAUSE
10: RUN
1120 PRINT AT 0,10;"EJEMPLO No.
";X;AT 0,10; OVER 1;"
": RANDOMIZE USR 61953: POKE N
UMSPR,N: RETURN
9999 CLEAR 47139: LOAD ""CODE 47
140,1888: LOAD ""CODE 61953,1495
: RUN
```

SPARES. Es una zona de trabajo donde se almacenarán los trozos de pantalla que van a ser borrados al dibujar los sprites, para poder restablecerlos después. La cantidad de memoria necesaria depende de los sprites que se vayan a utilizar y sus dimensiones. El valor que se le ha dado es un valor promedio que será suficiente para casi todos los casos. Si para vosotros resulta insuficiente, podéis colocar esta zona en otra parte de la memoria, cambiando el EQU de la etiqueta SPARES en el listado ensamblador.

En la dirección 54568 se encuentra la pantalla de memoria. Como sabéis, la pantalla del Spectrum tiene 256 píxels de ancho (32 caracteres) por 192 de alto, así que ocupa $32 \times 192 = 6144$ bytes (sin contar con los atributos), así que, en principio esto es lo que debería ocupar la pantalla de memoria. Sin embargo, esto no es así. Lo que pasa es que para poder generalizar al máximo la subrutina que se encarga de dibujar los gráficos en esta pantalla, necesitamos un carácter más de ancho, en el que se dibujarán restos de los sprites que, por estar a medias en la pantalla, no deben ser dibujados enteros. De esta manera, la pantalla de memoria ocupará $33 \times 192 = 6336$ bytes. En realidad, ocupa un byte más, porque también para generalización de la subrutina del dibujo, se necesita que esta columna adicional sea un píxel más alta que el resto de la pantalla. Así que en realidad, la zona ocupada por la pantalla comienza en 54567, pero la dirección que se corresponde con el inicio de la pantalla real del ordenador, es la 54568, que es la que tiene el nombre de ORIGSC.

A continuación de la pantalla, se encuentran sus atributos, en la dirección 60904, con la etiqueta ORIGAT. Al

igual que antes, necesitamos una columna más, con lo que ocuparán 33×24=792 bytes. En este caso, no necesitamos reservar especialmente otro byte antes de la dirección 60904, pues nos sirve el último de la zona de la pantalla, que será a la vez el último byte de la pantalla y el primero de los atributos.

En la dirección 61696 se encuentra la tabla de interrupciones. Se ha dicho muchas veces en esta y en otras revistas que en el modo 2 de interrupciones el microprocesador toma el contenido del registro I y el del bus de datos, que siempre es 255, y forma una dirección, de la que se extrae la dirección definitiva de la subrutina de interrupciones. Sin embargo, hay algunos periféricos que, por no estar demasiado bien hechos, hacen que el contenido del bus de datos en el momento de las interrupciones no sea 255. Esto quiere decir que la dirección donde el microprocesador buscará la dirección de las interrupciones, puede ser cualquiera cuyo byte alto sea el contenido del registro I. Para conseguir que sea cual sea el valor del bus datos, se lea la misma dirección para la subrutina de interrupciones, no hay más remedio que elegir para la subrutina una dirección cuyos byte alto y bajo sean iguales, y llenar con el valor de estos, 257 bytes a partir de la dirección XX00, siendo XX el contenido del registro I. En nuestro caso, para que la tabla de interrupciones esté en 61696, I valdrá 61696 / 256=241, y la subrutina de interrupciones comenzará en 62194=F2F2h, así que la tabla estará llena de 242=F2h. Es fácil ver que entonces, sea cual sea el contenido del bus de datos, la dirección formada con el registro I estará entre 61696 y 61951, y que el contenido de cualquiera de estas direcciones y de las siguientes será 242, con lo que siempre se saltará a la dirección 242 * 256+242=62194.

Tras la tabla de interrupciones tenemos, en 61953, INICIO, la subrutina que inicializa las interrupciones y crea la pantalla de memoria y las tablas de rotaciones.

En 62075 está DESACT la subrutina que desactiva las interrupciones y vuelve al modo normal.

En 62082 está la subrutina COMCHO, de comprobación de choque de sprites. En realidad no cabe entera aquí, pues se montaría sobre la dirección donde deben comenzar las interrupciones, por lo que parte de ella se encuentra al final de todo el bloque de la rutina de interrupciones y sus subrutinas.

En 62194 se encuentra la subrutina principal de interrupciones, junto con todas sus subrutinas y variables (excepto NUMSPR y SPRICH, que se encuentran en variables del sistema no usadas). Detrás, como ya hemos dicho, continúa la subrutina de comprobación de choque, que aunque no es muy larga, ha sido cortada en dos para aprovechar mejor la memoria, rellenando los huecos.

Y lo último que hay en la memoria son las ya mencionadas tablas de rotaciones, que comienzan en 63488. A propósito de estas tablas, seguramente os habréis preguntado qué sentido tiene almacenar un byte girado 0 veces (que es lo que hay en la primera tabla), puesto que el resultado será el mismo. La respuesta es bien sencilla. Si no incluyéramos esa tabla, sería necesario hacer dos subrutinas de dibujo, una para cuando el gráfico está en el primer píxel de un carácter y otra para el resto de los casos. El resultado quedaría menos elegante, sería más difícil de comprender y ocuparía prácticamente la misma memoria.

FUNCIONAMIENTO

El proceso que hay que realizar para mover los sprites por la pantalla es bastante complejo y lleva bastante tiempo. Por ello, lo vamos a dividir en dos partes, de tal modo que una vez haremos una parte, y la siguiente vez haremos la otra parte, y a la siguiente volveremos a hacer la primera, y así. Hacemos el volcado en pantalla de la pantalla de memoria, que es el proceso más lento, y la segunda, haremos el resto, que se puede dividir en las siguientes subetapas:

— Borrar los sprites de sus posiciones antiguas.

— Calcular nueva posición de los sprites y el gráfico que toque de los que compongan la animación de cada uno.

— Hacer todos los cálculos necesarios para el dibujo de los gráficos y su posterior borrado, almacenando los resultados en la tabla TABLDI, a la vez que se almacenan los trozos de pantalla que van a ser ocupados por los sprites.

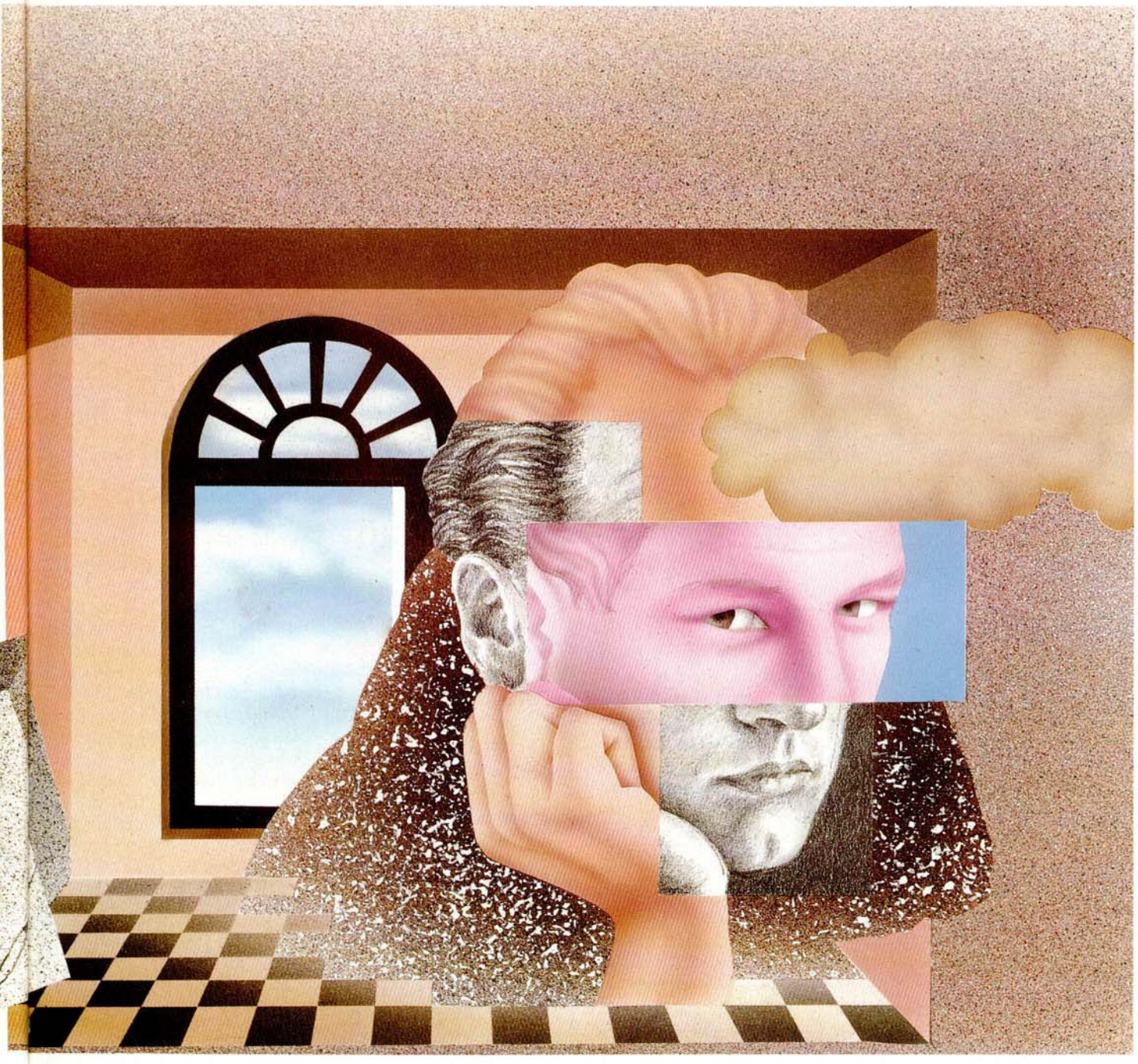
— Dibujar los sprites que se encuentren en la pantalla.

Ahora que sabemos todas las tareas a realizar, podemos comenzar a comentar el listado de la rutina.

El punto de entrada de ENTIT. Aquí, lo primero que hacemos es guardar todos los registros que vamos a utilizar. A continuación, estudiamos el contenido de la variable ESTADO. Si es 0, deberemos volcar la pantalla, y si es 1, hacer todo lo demás. En este segundo caso saltamos a NOVOLC. En el primero, llamamos a la subrutina que vuelca la pantalla y hacemos que ESTADO valga 1 para la próxima vez que se produzca una interrupción, para salir después por SALINT.

En el caso de encontrarnos en la segunda parte, ponemos ESTADO a 0 para la próxima vez. Después nos disponemos a borrar todos los sprites dibujados la última vez. Para ello, tomamos el contenido de NUBLBO, que excede en uno al número buscado. Antes de entrar en el bucle que borrará todos los gráficos, inicializamos IX con TABLDI, tabla que tiene los datos de los sprites que fueron dibujados y ahora queremos borrar, y HL con SPARES, donde están almacenados uno detrás de otro los trozos de pantalla necesarios para efectuar el borrado. Ahora, si el contenido de NUBLBO era 1, es que no habíamos dibujado ningún gráfico la última vez, y salimos del bucle sin haber entrado en él.

Ahora vamos, tomando de la tabla los datos necesarios. En DE cargamos la dirección en la pantalla de trabajo donde comenzará el proceso. Ahora, en vez de cargar en registros, hay unos datos que los vamos a meter en una dirección de memoria, de tal manera que luego serán recogidos directamente por una instrucción del tipo LD C,n, siendo n el dato en cuestión. Este sistema les resultará familiar a los que hayan leído los artículos sobre cómo se programa un juego, publicados en los MICROHOBBY 97-106, y escritos por el mismo autor. En BORPOI+1 y en BORPOD+1 guardamos la cantidad de bytes de ancho que hemos de restaurar. En AUMPOI+1 y en AUMPOD+1, guardamos 33 menos ese número, que será lo que hay que sumar a la dirección destino de la pantalla, tras restaurar un scan para lograr la dirección del siguiente. Por último, cargamos A con el número de scans que debemos restaurar. En BORPOI, aunque veamos un LD C,Ø estamos cargando C con el ancho del bloque a restaurar, pues es el valor que habíamos metido en BORPOI+1. Como en B teníamos Ø, en BC tenemos el número de bytes a renovar; en DE tenemos el destino, y el origen en HL, porque los trozos de pantalla que vamos a recuperar han sido anteriormente guardados en SPARES, así que tenemos los parámetros necesarios para hacer un LDIR. Ahora calculamos la dirección en panta-



lla del siguiente scan, sumándole al contenido de DE tras el 1dir, el número calculado antes y metido en AUMPOI+1 (por si no lo habéis notado, os diré que en la pantalla de trabajo, los scans están uno tras otro, a diferencia de cómo están en la pantalla del ordenador). Tras esto, cerramos el bucle para terminar con todos los scans. Ahora, nos queda restaurar los atributos. Para ello, cargamos en DE la dirección destino de atributos, y en A el número de filas que ocupa. Por lo demás, la restauración de los atributos es idéntica a la que acabamos de hacer. Tras restaurar por completo un trozo de pantalla, cerramos el bucle que los restaurará todos. Ya hemos terminado la primera de las subfases antes citadas.

Ahora continúamos por INSABO. Aquí hacemos un bucle en el que iremos moviendo todos los sprites y calculando los datos para la tabla TABLDI en los que estén en la pantalla. Dentro de este bucle, IY será el puntero para el elemento correspondiente al sprite dentro de la tabla de sprites TASPRI, IX el de TABLDI, y DE señalará la primera dirección libre de la zona SPARES, pues en este bucle también será donde guardaremos los trozos de pantalla que hagan falta. Dentro del bucle, lo que encontramos son llamadas a dos subrutinas. MOVERR se encargará de calcular las nuevas coordenadas del sprite, teniendo en cuenta si se mueve en línea recta o según trayectoria prefijada, y de calcular la fase de animación en que se encuentra, si la tiene. CREADA crea el elemento en la tabla TABLDI y guarda los trozos de pantalla.

Una vez guardados todos los trozos de pantalla, ya podemos dibujar los sprites en sus nuevas posiciones, llamando a DIBUJA, que se apoyará en los datos de TABLDI. Ahora, antes de volver, hacemos un rst 56 para leer el teclado, y después entramos en SALINT, donde se recuperan los registros y se vuelve. Nótese que tal y como está puesto, y teniendo en cuenta que la fase de volcado de pantalla abarca el tiempo de dos interrupciones, la lectura del teclado se hará tres veces menos de lo normal. Esto hace que la respuesta del mismo al teclear sea bastante pobre, pero da un poco más de velocidad al no tener que ejecutarse tantas veces la subrutina de lectura de teclado. Si pensamos usar los sprites desde Código Máquina, podría ser útil quitar las líneas 96, 97 y 98 para ahorrar aún más tiempo.

Y esto es todo el programa principal. Ahora quedan las subrutinas, que son cuatro: VUELCA, MOVERR, CREADA y DIBUJA.

VUELCA es la que se encarga de copiar la pantalla de trabajo en la pantalla del ordenador. Se trata del proceso que más tiempo consume, ya que hay que mover casi 7 K. La rutina vuelca todos los scans de arriba a abajo, y por cada 8, vuelca una fila de atributos. Puede pareceros que sería más sencillo volcar primero toda la pantalla sin atributos, y luego los atributos, pero el resultado sería que el movimiento quedaría mucho menos suave. Con este procedimiento habrá pequeñas zonas de la pantalla donde sprites parecerán doblarse; pero en la inmensa mayoría de los casos, la suavidad de movimientos será perfecta.

Veamos cómo funciona la subrutina. En los registros alternativos guardamos en HL el origen de pantalla de trabajo, y en DE el de la pantalla del ordenador, además de poner C a Ø. En los registros normales ponemos en HL el origen de los atributos de memoria, y en DE su correspondiente en la pantalla del ordenador. Asimismo, cargamos C con Ø y B con 25 (estas dos últimas operaciones de una sola vez. B será el contador del bucle que se repetirá para cada fila, dentro del cual deberemos volcar ocho scans y una fila de atributos. El bucle se repetirá 22 veces, a pesar de que inicialicemos B con 25. La razón es que, como veremos enseguida, a cada paso por el bucle, el registro BC será decrementado 32 veces, así que al final de las 22 veces que se repetirá el bucle, BC habrá sido decrementado en $22 \times 32 = 704$, con lo que B habrá sido decrementado en 3. Por tanto, deberemos empezar con B valiendo 3 más de las veces que se repita el bucle. Dentro del bucle de las 22 filas, volvemos a los registros alternativos para volcar los scans. Cargamos B con nueve para entrar en un bucle que se repetirá ocho veces, adivinad porqué. Dentro de este segundo bucle, trasladamos los 32 bytes que componen un scan de la pantalla desde la idem de memoria a la real. Posiblemente os sorprenda que para esto utilicemos 32 instrucciones LDI. Pero resulta que esto es mucho más rápido que cargar BC con 32 y hacer un LDIR, y además, así podemos utilizar el registro B como contador. Estos DLIs y los que veremos a continuación son la causa de que en un bucle tengamos que cargar B con 25 y en el otro 9 en vez de 22 y 8 respectivamente. Tras esto, calculamos en DE la dirección del siguiente scan. Dentro de este bucle siempre vamos a pasar de un scan a otro de la misma fila, y, como muchos ya sabréis, el algoritmo para pasar de un scan a otro de la misma fila es simplemente incrementar el byte alto de la dirección. Pero antes debemos restarle 32, porque al hacer los LDIs es como si le hubiéramos sumado 32. Como no hay registros dobles libres, nos valemos de A para hacer la resta. Primero le restamos 32 a E. Si no se nos produce acarreo, la resta ya está bien hecha y procedemos a

Fig. 6

TABLA DE DATOS DE LOS GRÁFICOS QUE SE VAN A DIBUJAR

Dirección relativa	Contenido
DIR+Ø	Dirección de destino en pantalla de memoria.
DIR+2	Dirección del gráfico.
DIR+4	Dirección de la máscara.
DIR+6	Número de scans.
DIR+7	Bytes de ancho.
DIR+8	Bytes que no entran en pantalla.
DIR+9	Byte de rotación.
DIR+1Ø	Byte de atributos.
DIR+11	Dirección de destino en atributos de memoria.
DIR+13	Número de filas de atributos que ocupan el gráfico.
DIR+14	Número de columnas de atributos.

incrementar D. Pero si hay acarreo, deberemos restarle uno a D, pero como después tendríamos que incrementarlo de nuevo, no hacemos ninguna de las dos cosas. Antes de cerrar el bucle, HL necesita ser incrementado para saltarnos la columna de más que ya dijimos que había en la pantalla de trabajo. Tras salir del bucle, en el que habremos volcado los ocho scans, hacemos los ajustes necesarios para pasar al primer scan de la siguiente fila, teniendo en cuenta que es posible que estemos pasando de un tercio a otro. No creo que estos ajustes merezcan más explicación, pues ya se han gastado muchos litros de tinta hablando de cómo se calcula el scan siguiente a uno dado en la pantalla del Spectrum. Ahora volvemos a los otros registros, y trasladamos una fila de atributos. Tras incrementar HL para saltarnos la columna sobrante, cerramos el bucle.

Llegamos ahora a MOVERR, la subrutina que calcula las nuevas coordenadas de cada uno de los sprites. A esta subrutina se llega con IY apuntando a los datos del sprite en tratamiento dentro de la tabla de sprites TASPRI. IX, por su parte, apuntará al lugar donde debemos crear el elemento correspondiente de la tabla TABLDI. Lo primero que hacemos al entrar es comprobar que el sprite esté activado, para lo cual, como ya hemos dicho, comprobamos que no sea 127 el sexto byte de la tabla (señalado por IY+5). Si es 127, regresamos inmediatamente. A continuación, guardamos en D el décimo byte de la tabla, que era el que indicaba el color, el tipo de

LISTADO 2

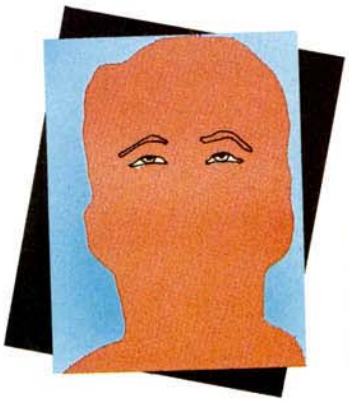
```
LÍNEA     DATOS                CONTROL

  1  7E84BD0400040000400004   463
  2  0004000400040000400004    20
  3  007EC4BD02FE02FE02FE02FE 1279
  4  02FE02FE02FE02FE02FE00FC 1280
  5  7E04BE00FC00FC00FC00FC7E 1076
  6  FC00FC00FC00FC00FC00FC7E 1386
  7  44BEFEFEFEFEFEFEFEFE7E84 2290
  8  FEFEFEFEFEFEFEFEFEFEFEFE 2290
  9  BEFC00FC00FC00FC00FC00FC 1450
 10  00FC00FC00FC007EC4BE00FC 1268
 11  FE02FE02FE02FE02FE02FE02 1280
 12  FE02FE02FE027E04BF00FE00 1089
 13  0400040004000400040004   20
 14  0400040004047E44BF020202 401
 15  0400040004047E44BF020202 20
 16  0202020202020202020202   135
 17  0202020207F0000000000000 128
 18  0000000000000000008000   1395
 19  5C00BFF05FFEBFF05C00      128
 20  8000000000000000000000   1912
 21  0000FFFFFFFFFFFF7FFF     307
 22  23FF000F000100000001     1707
 23  000F23FF7FFFFFFFFFFF     1020
 24  FFFFFFFF000000000000     1146
 25  00FF0003FFC007C3E00F     932
 26  3CF01EC3781D00B83A10     482
 27  5C3A105C34102C34102C     416
 28  34102C34002C3A005C3A     845
 29  005C3D00BC3EC37C3F3C     1806
 30  FC27C3E427FFE43FFFFC     1020
 31  0000000000000FFFFFFFFF   1089
 32  00FFFC003FF8001FF000     636
 33  0FE00007C00003C00003     515
 34  8000018000018000018000   387
 35  0001800001800001800001   388
 36  0180000180000180000180   515
 37  8000018000018000FF0000   895
 38  0001800001FFFFFF00000    705
 39  0000000000FF0003FFC0     1115
 40  07C3E00F3CF01EC3781D     590
 41  00B83A105C3A125C3414     372
 42  2C34182C34102C34002C     611
 43  3A005C3A005C3D00BC3E     1450
 44  C37C3F3CFC27C3E427FF     798
 45  E43FFFFC000000000000     1838
 46  FFFFFFFF00FFFC003FF8     709
 47  001FF0000FE00007C000     456
 48  03C0000380000180000180   515
 49  8000018000018000018000   387
 50  0001800001800001800001   388
 51  0180000180000180001FF    642
 52  8000018000000000000FF    765
 53  FFFF000000000000003CF0   1191
 54  0003FFC007C3E00F3CF0     782
 55  1EC3781D00B83A105C3A     415
 56  105C34102C34102C341F     440
 57  2C34002C3A005C3A005C     1044
 58  3D00BC3EC37C3F3CFC27     1515
 59  C3E427FFE43FFFFC00FF     1275
 60  00000000FFFFFFFF00FE0    1073
 61  FC003FF8001FF0000FE0     525
 62  0007C00003C000038000     388
 63  0180000180000180000180   515
 64  8000018000018000018000   387
 65  0001800001800001800001   388
 66  0180000180000180001FF    642
 67  8000018000000000000FF    765
 68  FFFF000000000000003CF0   1191
 69  0003FFC007C3E00F3CF0     782
 70  1EC3781D00B83A105C3A     415
 71  105C34102C34102C341F     440
 72  2C34002C3A005C3A005C     1044
 73  3D00BC3EC37C3F3CFC27     1515
 74  C3E427FFE43FFFFC00FF     1275
 75  00000000FFFFFFFF00FE0    1073
 76  FC003FF8001FF0000FE0     525
 77  0007C00003C000038000     388
 78  0180000180000180000180   515
 79  8000018000018000018000   387
 80  0001800001800001800001   388
 81  0180000180000180001FF    642
 82  8000018000000000000FF    765
 83  FFFF000000000000003CF0   1191
 84  0003FFC007C3E00F3CF0     782
 85  1EC3781D00B83A105C3A     415
 86  105C34102C34102C341F     440
 87  2C34002C3A005C3A005C     1044
 88  3D00BC3EC37C3F3CFC27     1515
 89  C3E427FFE43FFFFC00FF     1275
 90  00000000FFFFFFFF00FE0    1073
 91  FC003FF8001FF0000FE0     525
 92  0007C00003C000038000     388
 93  0180000180000180000180   515
 94  8000018000018000018000   387
 95  0001800001800001800001   388
 96  0180000180000180001FF    642
 97  8000018000000000000FF    765
 98  FFFF000000000000003CF0   1191
 99  0003FFC007C3E00F3CF0     782
100  1EC3781D00B83A105C3A     415
101  105C34102C34102C341F     440
102  2C34002C3A005C3A005C     1044
103  3D00BC3EC37C3F3CFC27     1515
104  C3E427FFE43FFFFC00FF     1275
105  00000000FFFFFFFF00FE0    1073
106  FC003FF8001FF0000FE0     525
107  0007C00003C000038000     388
108  0180000180000180000180   515
109  8000018000018000018000   387
110  0001800001800001800001   388
111  0180000180000180001FF    642
112  8000018000000000000FF    765
113  FFFF000000000000003CF0   1191
114  0003FFC007C3E00F3CF0     782
115  1EC3781D00B83A105C3A     415
116  105C34102C34102C341F     440
117  2C34002C3A005C3A005C     1044
118  3D00BC3EC37C3F3CFC27     1515
119  C3E427FFE43FFFFC00FF     1275
120  00000000FFFFFFFF00FE0    1073
121  FC003FF8001FF0000FE0     525
122  0007C00003C000038000     388
123  0180000180000180000180   515
124  8000018000018000018000   387
125  0001800001800001800001   388
126  0180000180000180001FF    642
127  8000018000000000000FF    765
128  FFFF000000000000003CF0   1191
129  0003FFC007C3E00F3CF0     782
130  1EC3781D00B83A105C3A     415
131  105C34102C34102C341F     440
132  2C34002C3A005C3A005C     1044
133  3D00BC3EC37C3F3CFC27     1515
134  C3E427FFE43FFFFC00FF     1275
135  00000000FFFFFFFF00FE0    1073
136  FC003FF8001FF0000FE0     525
137  0007C00003C000038000     388
138  0180000180000180000180   515
139  8000018000018000018000   387
140  0001800001800001800001   388
141  0180000180000180001FF    642
142  8000018000000000000FF    765
143  FFFF000000000000003CF0   1191
144  0003FFC007C3E00F3CF0     782
145  1EC3781D00B83A105C3A     415
146  105C34102C34102C341F     440
147  2C34002C3A005C3A005C     1044
148  3D00BC3EC37C3F3CFC27     1515
149  C3E427FFE43FFFFC00FF     1275
150  00000000FFFFFFFF00FE0    1073
151  FC003FF8001FF0000FE0     525
152  0007C00003C000038000     388
153  0180000180000180000180   515
154  8000018000018000018000   387
155  0001800001800001800001   388
156  0180000180000180001FF    642
157  8000018000000000000FF    765
158  FFFF000000000000003CF0   1191
159  0003FFC007C3E00F3CF0     782
160  1EC3781D00B83A105C3A     415
161  105C34102C34102C341F     440
162  2C34002C3A005C3A005C     1044
163  3D00BC3EC37C3F3CFC27     1515
164  C3E427FFE43FFFFC00FF     1275
165  00000000FFFFFFFF00FE0    1073
166  FC003FF8001FF0000FE0     525
167  0007C00003C000038000     388
168  0180000180000180000180   515
169  8000018000018000018000   387
170  0001800001800001800001   388
171  0180000180000180001FF    642
172  8000018000000000000FF    765
173  FFFF000000000000003CF0   1191
174  0003FFC007C3E00F3CF0     782
175  1EC3781D00B83A105C3A     415
176  105C34102C34102C341F     440
177  2C34002C3A005C3A005C     1044
178  3D00BC3EC37C3F3CFC27     1515
179  C3E427FFE43FFFFC00FF     1275
180  00000000FFFFFFFF00FE0    1073
181  FC003FF8001FF0000FE0     525
182  0007C00003C000038000     388
183  0180000180000180000180   515
184  8000018000018000018000   387
185  0001800001800001800001   388
186  0180000180000180001FF    642
187  8000018000000000000FF    765
188  FFFF000000000000003CF0   1191
189  0003FFC007C3E00F3CF0     782
     1EC3781D00B83A105C3A     624
     905C34502C343302C3410    440
     2C34002C3A005C3A005C
```

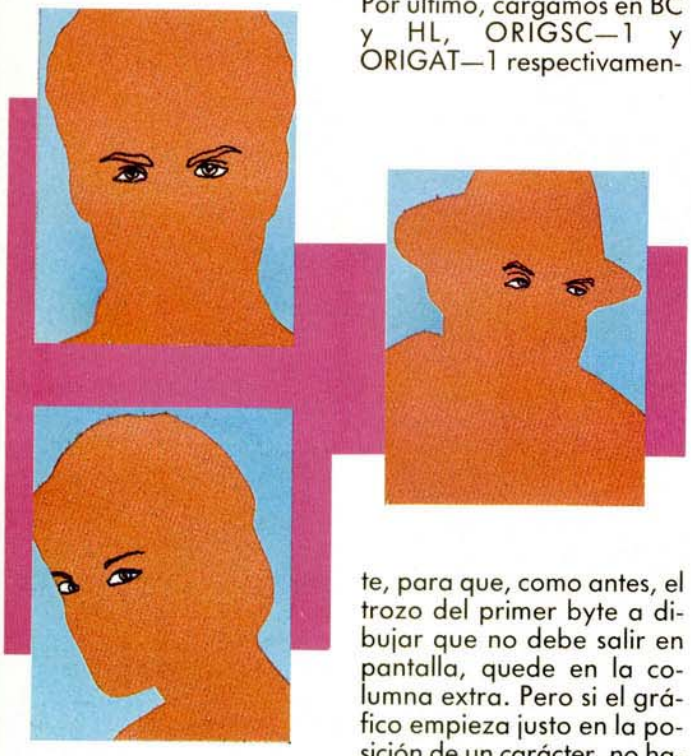
DUMP: 40.000
N.° BYTES: 1.888

animación y el tipo de movimiento. Aislamos el color y lo guardamos en su lugar correspondiente en la tabla TABLDI. Ahora, si el movimiento es en línea recta, saltamos a RECLIN. Si es en trayectoria prestablecida, cargamos en HL la dirección de los datos de la trayectoria. Leemos el valor contenido en esa dirección. Si es 127, la trayectoria ha finalizado y cargamos en HL la dirección inicial para volverla a repetir. Tanto si hacemos esto como si no, continúamos por NOFITA. Ahora comprobamos si el valor leído es un 126. En este caso, los dos siguientes bytes indicarán la nueva dirección inicial de los gráficos correspondientes al sprite, y los pasamos a su lugar en la tabla TASPRI, para retroceder a continuación a RECUPE y leer un nuevo dato de la tabla de datos de la trayectoria. Cuando nos encontramos con un dato que no es 127 ni 126, llegamos a TRANOR. Este dato será el incremento X, y lo pasamos a E, cargando A el siguiente valor que será el incremento Y, tras actualizar el contador de trayectoria, saltamos a COCORE, donde nos reuniremos con la bifurcación hecha en el caso de un movimiento en línea recta. En este casò saltábamos a RECLIN. Aquí, cargamos en E el incremento X y en A el Y, para entrar en COCORE con los mismo datos que si hubiéramos venido desde el caso de trayectoria prefijada. En COCORE, pasamos el incremento Y a BC. Hay que tener en cuenta que los incrementos son números con signo, y al pasarlos a un registro doble, el registro bajo deberá ser igual que el byte de incremento, pero el alto, deberá ser Ø para un incremento positivo y 255 para un incremento negativo. Esto lo conseguimos con el CP 128 y el CCF, que pone el banderín de acarreo alzado para un número negativo, y el SBC A,A, que dejará en A un Ø si el acarreo estaba bajo, y un —1 (un 255) si estaba alto. Este incremento, una vez convertido en un número de 16 bits, se lo sumamos a la coordenada Y, y guardamos la nueva coordeanda. A continuación, hacemos lo propio con la X. Ya hemos actualizado las coordenadas. Ahora vamos a pasar a calcular qué gráfico hay que dibujar, de los varios que puede tener un sprite, para lo cual tendremos que tener en cuenta la fase de animación. Por eso cargamos en E el número total de fases de animación y en A la fase en la que estaba la última vez. Ahora saltamos a ADETRA si la animación es del tipo adelante-atrás. Si es cíclica, simplemente incrementamos la fase, y si hemos llegado a la última, la ponemos a Ø. Después saltamos a TRAREN. Nótese que en la animación cíclica, coinciden la fase actual con el gráfico que hay que dibujar. Esto quiere decir que si la fase es Ø habrá que dibujar el primer gráfico, si es 1, el segundo, etc. En ADETRA tratamos la animación de adelante-atrás. En este tipo de animación, si por ejemplo, tenemos cinco gráficos distintos para la animación, primero habrá cinco fases que coincidirán con las de la cíclica. Pero después, habrá tres más en las que se dibujarán los tres gráficos del centro en orden inverso: primero el cuarto, luego el tercero, y después el segundo. En general, si hay N gráficos distintos, la cantidad de fases por las que pasaremos será de 2*N-2. Por eso ADETRA, tras incrementar el contador de fase, le restamos ese número. Si el resultado es Ø, el ciclo habrá terminado y comenzaremos de nuevo por la fase Ø. Pero antes hemos guardado el resultado de la resta. Ahora tenemos que calcular qué es el que hay que dibujar. Si la fase actual es menor que la cantidad de gráficos distintos que hay, nos encontramos en la parte que es igual que en la animación cíclica, por lo que la fase es igual al número del gráfico. En caso contrario, bastará que calculemos 2*N-2-fase actual, pero eso es lo que habíamos calculado antes, salvo que con signo contrario así que en PARETR lo recuperamos (había sido guardado en PARETR+1), lo cambiamos de signo y continuamos por TRAREN como en los otros casos. En TRAREN, cargamos DE con la cantidad de memoria que ocupa cada gráfico, y la multiplicamos por dos caras para calcular la que ocupa cada gráfico con su mascara. Este es el valor que le tendremos que sumar a la dirección del primer gráfico del sprite tantas veces como indique el contenido de A. Tras hacer esto, guardamos la nueva dirección del gráfico en el lugar que le corresponde en la tabla TABLDI. Y esto es todo lo que hace MOVERR.

Llegamos ahora a CREADA, posiblemente, la subrutina más complicada. Básicamente, lo que se hace en ella es averiguar si hay que dibujar todo el gráfico, o éste se encuentra parcialmente en la pantalla, y calcular todos los datos para que luego DIBUJA se encargue ya de dibujar los gráficos. Lo primero que hacemos es calcular qué tabla de rotaciones de las cuatro existentes es la que vamos a utilizar. En realidad, lo que calculamos es el byte alto de su dirección de inicio (el bajo es Ø). Tras guardar esto en su lugar, comprobamos si la coordenada X es positiva y menor de 256. En este caso saltamos a XPOSIT. En caso contrario, si no es mayor de —256, retornamos inmediatamente, pues el sprite se encuentra fuera de la pantalla. Cuando es negativa mayor que —256, comprobamos si es mayor o igual que —6, en cuyo caso, el trozo del gráfico que no se ve en pantalla será menor de un carácter. Esto quiere decir que habrá que dibujar el gráfico entero, pero empezando en la columna sobrante de la pantalla de memoria, para que quede aquí el trozo que no se verá en pantalla. Por eso cargamos en BC ORIGSC—1 y en HL ORIGAT—1. Es en estos registros donde vamos a construir las direcciones destino del gráfico en la pantalla de trabajo y en los atributos de la misma. Como hemos dicho que dibujaremos el gráfico entero, hace-



mos que el ancho del gráfico en la pantalla sea igual al ancho del gráfico en sí, y que el número de bytes de ancho que no entran en la pantalla es Ø. Tras esto saltamos a PARTEY para hacer cálculos similares con la coordenada Y. Continuamos en NEGARE, a donde llegamos cuando la coordenada X es mayor que —256 pero menor que —6. Como las dimensiones de los gráficos en la magnitud X viene dada en caracteres, vamos a pasar esta coordenada X de alta a baja resolución, para lo cual la dividimos por 8, teniendo en cuenta que se trata de un número negativo, y por lo tanto, los bits que entren



por la izquierda deben ser unos y no ceros. A esta coordenada en baja resolución, le sumamos el ancho del gráfico. Si el resultado es cero o ni siquiera llega, es porque el sprite está completamente fuera de la pantalla, y retornamos enseguida. Si el resultado excede de cero, coincidirá con la cantidad de caracteres de ancho que del gráfico cabrán en la pantalla, y lo guardamos en su lugar en TABLDI. Ahora si que hay un trozo del gráfico que no se dibuja por estar fuera de la pantalla. El ancho de este trozo será igual a la coordenada en baja resolución del sprite, sólo que cambia de signo. Pero para esos bytes que no se van a dibujar sean tomados de la derecha, como debería ser, y no de la izquierda, este número no sólo lo guardaremos en IX+8, sino que se lo sumaremos a la dirección de comienzo del gráfico. Por último, cargamos en BC y HL, ORIGSC—1 y ORIGAT—1 respectivamente, para que, como antes, el trozo del primer byte a dibujar que no debe salir en pantalla, quede en la columna extra. Pero si el gráfico empieza justo en la posición de un carácter, no habrá ningún byte que quede a medias entre dos direcciones, o lo que es lo mismo, el primer byte a dibujar del gráfico quedará completamente dentro de la pantalla, por lo que hacemos que BC y HL incrementen en uno sus valores antes de saltar a PARTEY. Seguimos ahora por XPOSIT. Aquí la coordenada X estará entre Ø y 255, y pueden ocurrir dos casos: que el sprite quede totalmente dentro o que se salga parte por la derecha. Vamos a comprobarlo. Si la X es Ø, consideramos desde el principio que el sprite está totalmente dentro de la pantalla. En caso contrario, calculamos la columna en baja resolución donde terminaría, el dibujo. Si no llega a 33, el gráfico cabe entero. En caso contrario, al valor obtenido le restamos 32 y ya tenemos cuantos bytes de ancho no tienen que ser dibujados. Lo restamos del ancho del sprite y obtenemos cuántos han de serlo. En RECOIN calculamos la dirección destino en la pantalla de trabajo y saltamos a PARTEY. En INTEGE, lo único que hacemos es indicar que el ancho del gráfico que cabe en pantalla es igual al ancho total de éste y que el ancho de la parte que no cabe es Ø, y después retrocedemos a RECOIN para calcular la dirección de la pantalla de trabajo. Y llegamos a PARTEY. Aquí se hace lo mismo que acabamos de hacer con la X, así que no la vamos a comentar tan en detalle, sino sólo a ver las diferencias. Aquí, cuando un sprite no está completamente en la pantalla, no necesitamos calcular nada más que cuantos scans quedan dentro, y no cuántos fuera, aunque cuando se sale por arriba, igual que cuando con la X se salía por la izquierda, habrá que modificar la dirección de comienzo del gráfico, sumándole el ancho del gráfico tantas veces como scans quedan fuera de la pantalla. Lo único que se hace y que no se hacía con la X es calcular cuántas filas de atributos ocupa el gráfico. En el caso de un gráfico que se sale por arriba o por abajo, cogemos el número de scans que caben en pantalla, lo dividimos por 8, y si no resultado exacto, al cociente le sumamos 1 y nos olvidamos del resto, y ese cociente será lo que buscábamos. En el caso de un sprite que cabe completamente en la pantalla, se calcula la fila de la pantalla dentro de la que está el primer scan del gráfico, y la fila dentro de la que esta el último, se restan, se le suma 1 y obtenemos el resultado deseado. Continuaremos ahora la explicación más detalladamente desde COYPOS. En primer lugar, guardamos en su lugar correspondiente en la tabla la dirección de la pantalla de memoria a la que va a ir el gráfico. A continuación calculamos la dirección de la máscara a partir de la dirección del gráfico y de la memoria ocupada por un gráfico. Ahora calculamos cuántas columnas de atributos ocupa el gráfico, si está en el principio de una columna, ocupará tantas columnas como su ancho en caracteres, pero en caso contrario, ocupará una más. Ahora recuperamos de la pila a DE y HL (que han sido guardados en el fragmento de listado que no hemos visto con detenimiento), y volvemos a guardar HL. Tras meter la dirección de atributos en su sitio y aumentar en 1 el contenido de la variable NUBLBO para indicar que hay un sprite más para dibujar, llegamos a las líneas que se encargan de guardar en SPARES (recuérdese que DE apunta a SPARES) cada uno de los trozos de pantalla donde luego van a ir los sprites. Tampoco necesitan comentarios, pues son prácticamente idénticas a las que tomaban estos trozos de SPARES y los copiaban en la

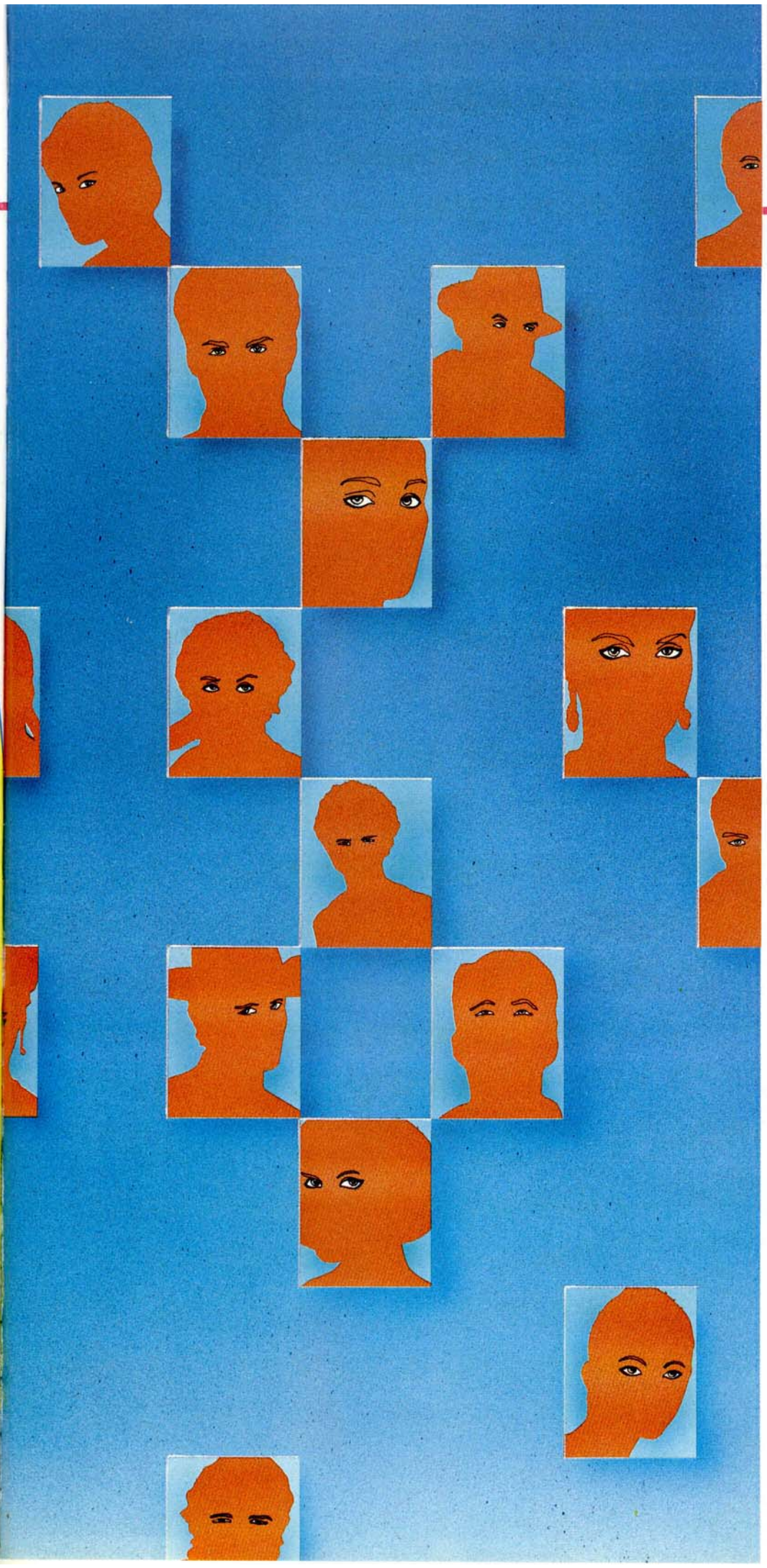
pantalla. Al final, actualizamos IX para que apunte al siguiente elemento de la tabla.

Y llegamos a DIBUJA, la subrutina que, utilizando el gráfico y la máscara y todos los datos de la tabla TABLDI, se encarga de efectuar los dibujos de los sprites en la pantalla de memoria. El mayor problema de comprensión que puede presentar esta subrutina es que maneja demasiados datos y puede uno perderse. Como contador para el bucle, que se repetirá tantas veces como sprites haya que dibujar, utilizamos A guardado en la pila. Veamos todos los datos que cargamos dentro de este bucle antes de proceder a efectuar el dibujo. En DE cargamos la dirección de la pantalla de trabajo. En H, el byte alto de la tabla de rotación. En PONUDO+1, el valor 255 girado N veces (siendo N el número de pixels que debemos desplazar todo el gráfico antes de dibujarlo). En POSNUM+1, almacenamos un 255 girado (8-N) veces hacia la izquierda. Ahora intercambiamos los registros con los alternativos. En HL, cargamos la dirección del gráfico para luego pasarla a IY. En HL cargamos la dirección de la máscara. En POSBYT+1, cargamos el ancho en caracteres. En SUMVAL+1, cargamos 33 menos el número de scans de ancho. Este es el valor que hay que sumarle a la dirección de pantalla tras haber dibujado un scan para obtener la dirección del siguiente. C será el contador de los scans que debemos dibujar. Por

LISTADO 3

LÍNEA	DATOS	CONTROL
1	F33EF1ED4705002100F1	1134
2	36F22310FB36F21128D5	1164
3	2100403EC008E5012000	621
4	EDB013E1247CE607200A	1096
5	7DC6206F38047CD60867	975
6	083D20E3210005811E8ED	935
7	3E1801200EDB0133D20	644
8	F7ED5E0E0426F83E0991	1098
9	912E00451E0057152806	444
10	CB38CB1B18F770247325	1060
11	2D20EC24240D20E13E01	718
12	329DF7AF32815C329CF7	1353
13	FBC93E3FED56ED47C9ED	1646
14	4BB05C0C280A0DCDB3F2	1044
15	010000D003C9DD21B05C	935
16	3A815CDD340008ED4BB0	1048
17	5C78B92807CDB3F20101	1072
18	00D8083D20E90BC921B9	980
19	F722B7F7219EF722EAF2	1659
20	CDCFF2D021C0F722B7F7	1798
21	21AAF722EAF278CDB3F7	1711
22	EB79CDB3F7A7ED527830	1641
23	087C2F677D2F6F2379CD	926
24	C7F7C3000000000000000	641
25	00F3F5C5D5E5DDE5D908	1802
26	F5C5D5E5FDE53A9CF7A7	1994
27	C213F3CDAFF33E01329C	1348
28	F7C39EF32128D1AF329C	1506
29	F73A9DF7DD2129D03D28	1313
30	4A08DD5E00DD5601DD7E	1052
31	0E3244F3325AF3ED44C6	1261
32	213248F3325EF3DD7E06	1138
33	06000E00EDB00E00EB09	691
34	EB3DC243F3DD7E0DDD5E	1475
35	0BDD560C0E00EDB00E00	771
36	EB09EB3DC259F30E0FDD	1316
37	0908C321F33E01329DF7	1005
38	1128D1DD2129D0FD2108	1063
39	CF3A815C47A72814C5D5	1194
40	CD5FF4D1CD1BF5011100	1248
41	FD09C110EFCDE5F6FDE1	1868
42	FDE5FFFDE1E1D1C1F108	2091
43	D9DDE1E1D1C1F1FBED4D	2096
44	2128D51100400E00D921	631
45	E8ED11005801001 9D906	823
46	09EDA0EDA0EDA0EDA0ED	1834
47	A0EDA0EDA0EDA0EDA0ED	1985
48	A0EDA0EDA0EDA0EDA0ED	1985
49	A0EDA0EDA0EDA0EDA0ED	1985
50	A0EDA0EDA0EDA0EDA0ED	1985
51	A0EDA0EDA07BD6205F38	1474
52	A0EDA0EDA07BD6205F38	758
53	01142310B67BC6205F38	1446
54	047AD60857D9EDA0EDA0	1985
55	EDA0EDA0EDA0EDA0EDA0	1985
56	EDA0EDA0EDA0EDA0EDA0	1985
57	EDA0EDA0EDA0EDA0EDA0	1985
58	EDA0EDA0EDA0EDA0EDA0	1985
59	EDA0EDA0EDA0EDA0EDA0	1985
60	EDA0EDA0EDA0EDA0EDA0	1509
61	2305C2C1F3C9FD7E05FE	1443
62	7FC8FD7E0957E647DD77	1131
63	0ACB622831FD6E0DFD66	1186
64	0E7EFE7F2007FD6E0AFD	1079
65	660B7EFE7E200E237EFD	1003
66	7700237EFD770123C378	1287
67	F45F237E23FD750DFD74	1486
68	0EC3AAF4FD7E0A5FFD7E	1134
69	0B4FFE803F9F47FD6E06	1123
70	FD660709FD7506FD7407	1244
71	7B4FFE803F9F47FD6E04	1117
72	FD660509FD7504FD7405	1464
73	FD7E0CFD5E08CB5AC2E7	1278
74	F43CBB2001AFFD770CC3	1438
75	FDF43C579393C60232FA	1038
76	F428017AFD770CBB3804	1084
77	3E00ED44FD5E0FFD5610	1178
78	CB23CB12FD6E00FD6601	916
79	A7280447191 0FDDD7502	1239
80	DD7403C9FD7E044FE606	1386
81	C6F8DD7709FD7E05A728	985
82	4D3CC079C606301321E7	1218
83	ED0127D5FD7E02DD7707	1445
84	AFDD7708C3C1F5371FCB	1386
85	2FCB2F47FD8602D0C8DD	1254
86	770778ED44DD7708DD86	993
87	02DD77023E00DD8E03DD	1227
88	77037921E7ED0127D5E6	1121
89	06204D2303C3C1F579D6	1162
90	013836CB3FCB3FCB3FFD	1106
91	8602FE213829D620DD77	1170
92	08ED44FD8602DD770779	1379
93	CB3FCB3FCB3F4F21E8ED	1140
94	856F8C95677901 28D581	1445
95	4F889147C3C1F5FD7E02	1462
96	DD7707AFDD7708C398F5	1283
97	FD7E07A7283B3CC0FD7E	1387
98	06FD8603D0C8DD7706ED	1454
99	44FD8603E5D5DD6E02DD	756
100	660316 00FD5E02193DC2	1502
101	E6F5DD7502DD7403DD7E	1230
102	06C607CB3FCB3FCB3FDD	1522
103	770DC376F6FD7E06FEC0	1237
104	D0CB3FCB3FCB3FC50121	827
105	003C3D280409C313F6C1	1078
106	E5D5606911 2100FD7E06	925
107	A728071 93DC228F6444D	1322
108	FD7E06FD8603382CFEC1	945
109	3028FD7E03DD77067908	1370
110	FD4E06CB39CB39CB39FD	1196
111	7E06FD8603C607CB3FCB	1109
112	3FCB3F91DD770D084FC3	1373
113	76F63EC0FD9606DD7706	1343
114	C607CB3FCB3FCB3FDD77	1014
115	0DDD7100DD7001DD6E02	1068
116	DD6603FD5E0FFD561019	1219
117	DD7504DD7405DD5607DD	781
118	7E09E606280114DD720E	1457
119	D1E1E5DD750BDD740C60	1472
120	69DD7E0E32C8F632D6F6	1550
121	ED44C62132CCF632DAF6	1329
122	3A9DF73C329DF7DD7E06	517
123	06000E00EDB00E000093D	1290
124	20F6E1DD7E0D0E00EDB0	987
125	0E00093DC2D5F60E0FDD	1235
126	09C9DD2129D03A9DF73D	1391
127	C8F5DD5E00DD5601DD66	1017
128	092EFF7E3255F7247E25	1216
129	322BF7D9DD6E02DD6603	1591
130	E5FDE1DD6E04DD6605DD	1063
131	7E07322FF7ED44C62132	988
132	5BF7DD4E06DD5E081600	133
	D90100000D90600FD7E00	820
	FD23087E23D96F7EB124	1124
	4E086F7846258B66F08EB	960
	A6B37723EBD910E119FD	1470
	19D979F600EBA6B0773E	1367
	00856F8C9567EBD90DC2	1295
	29F7DD6E0BDD660CDD7E	1312
	0E3284F7ED44C6213280	1170
	F7DD5E0A16B8DD7E0D08	1146
	06007EA2B3772310F90E	906
	0009083DC282F7F10E0F	919
	DD09C3ECF60001FD5E02	1257
	CB23CB23CB23C3ADF7FD	1582
	5E031600A7ED52C9CDC7	1210
	F7C30000FD6E04FD6605	1169
	C9FD6E06FD6607C9FD21	1419
	08CFA7C8D5111100FD19	1107
	3D20FBD1C900000000000	754

DUMP: 40.000
N.º BYTES: 1.49



último, en DE cargamos el número de caracteres de ancho que no vamos a dibujar del sprite. Sobre LOSCAN se cerrará el bucle para cada scan.

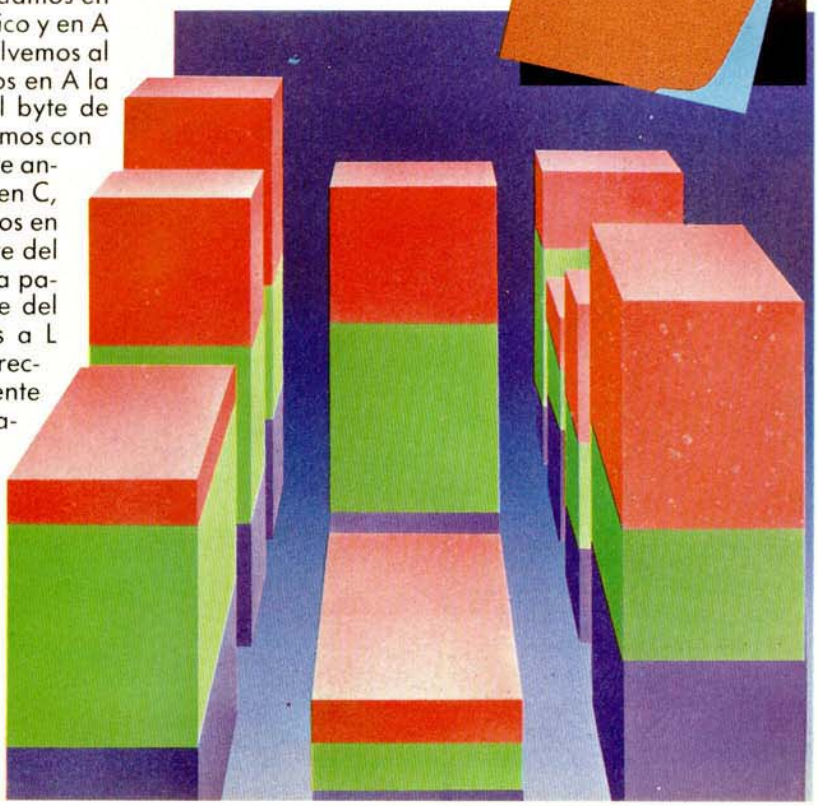
Aquí intercambiamos los registros. Ahora tenemos que tener bien claro cuál es el proceso a seguir para dibujar cada scan. Cada byte del gráfico tendrá que ser colocado entre dos direcciones de memoria dentro de la pantalla. Por tanto, en una dirección de la pantalla entrarán fragmentos de dos bytes del gráfico. Sabiendo que la tabla de rotaciones nos da por un lado, la parte del byte del gráfico que va en la dirección de pantalla actual y por otro lado la parte irá en la misma dirección que la primera parte del siguiente byte de gráfico, el proceso para cada byte sería: obtener la primera parte del byte. Juntarla con la segunda parte del anterior byte. Almacenar el resultado en pantalla. Obtener la segunda parte del byte y guardarla para su uso con el byte siguiente. En realidad, sería un poco más complicado, pues hay que hacer esto para el byte del gráfico y para el de la máscara. El paso descrito como guardar el resultado en pantalla, será en realidad hacer un AND de lo contenido en pantalla con el resultado de la máscara, a continuación hacer un OR con el resultado del gráfico y por último almacenar el resultado en pantalla. A partir de ahora, para intentar clarificar un poco las cosas, vamos a llamar B1, C1, etc., a los registros del juego que hemos inicializado primero, es decir, en el que DE tiene la dirección de la pantalla y HL la de la tabla de rotaciones, y B2, C2, etc., a los del otro juego, es decir, en el que HL contine la dirección de la máscara

y DE el número de bytes que no hay que dibujar. Para guardar la segunda parte del byte del gráfico, utilizaremos B1 para el gráfico y C1 para la máscara. Pero en el caso del primer byte de un scan, estos registros no podrán contener la segunda parte del byte anterior, porque éste no existe. Por esto, lo que hacemos es invertarnos un byte anterior, que para que no afecte al dibujo debe ser Ø para el gráfico y 255 (una vez girado y tomada su segunda parte) para la máscara.

Éstos son los valores que debemos cargar en B1 y C1 antes de entrar en el bucle para todos los bytes de un scan, y lo hacemos en POSNUM (recuérdese que previamente habíamos cargado en POSNUM+1 la segunda parte del resultado de rotar 255). Ahora volvemos al juego de registros 2, e inicializamos B2 que va a ser el contador del bucle para los bytes de un scan (el número de scan lo habíamos puesto en POSBYT+1). Sobre LOBYTS se cerrará el bucle. Aquí, guardamos en A' el byte del gráfico y en A el de máscara. Volvemos al juego 1. Cargamos en A la primera parte del byte de máscara y la juntamos con la segunda del byte anterior que estaba en C, y después cargamos en C la segunda parte del byte actual. Ahora pasamos a A el byte del gráfico y después a L para calcular la dirección correspondiente en la tabla de rotación. Como hemos incrementado H, HL estará apuntando a la segunda parte del byte, y no a la primera.

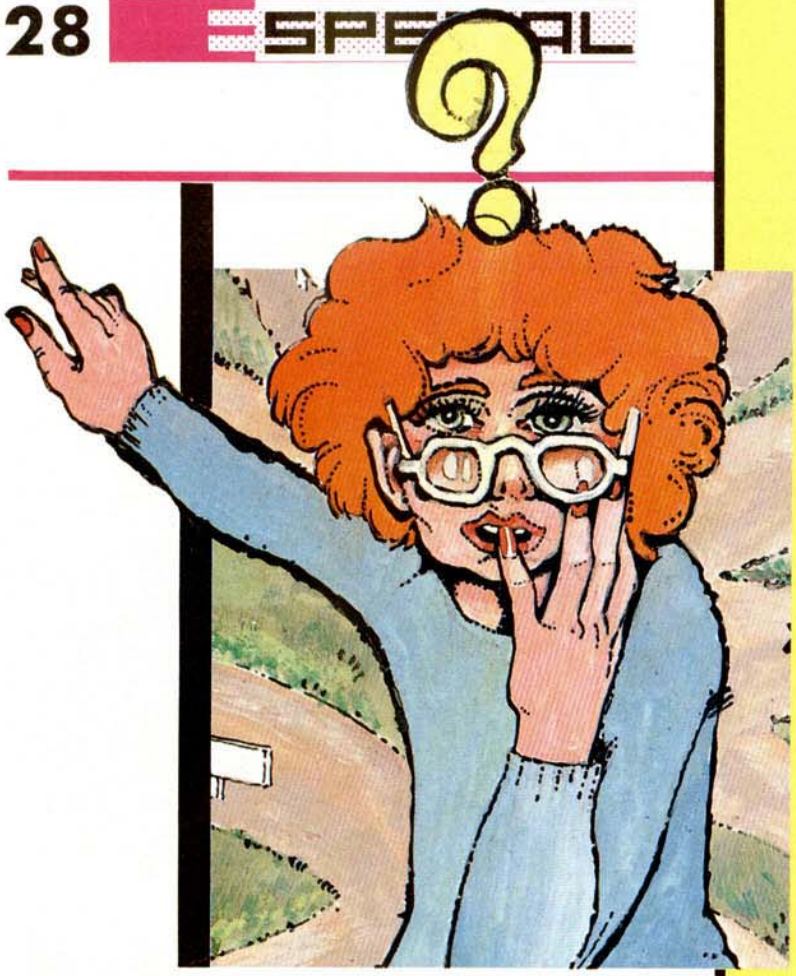
Cargamos en A la segunda parte del byte anterior y en B la de éste para usarlo la próxima vez.

Ahora decrementamos H y juntamos la segunda parte del byte anterior, que está en A con la primera de éste. Lo guardamos temporalmente en L y obtenemos la máscara, hacemos el AND con la pantalla, el OR con el gráfico y lo guardamos en la pantalla. Ahora volvemos al juego 2 para cerrar el bucle de cada byte de un scan. Sumanos DE a HL e IY para saltarnos la parte del gráfico que no ha de ser dibujada y de nuevo al juego 1. Ahora tenemos que dibujar la segunda parte del último byte del scan que no ha sido dibujada dentro del bucle. Al igual que como ocurría con el primer scan, con este último debemos inventarnos un byte siguiente, del que el gráfico sea Ø y la máscara 255. En PONUDO juntamos la primera parte del imaginario byte de máscara (colocada anteriormente en PONUDO+1) con la segunda del byte que ha quedado sin dibujar. Hacemos el AND con la pantalla y el OR con el gráfico y lo metemos en pantalla. Ahora le sumamos a la dirección de pantalla el valor necesario para calcular la dirección del siguiente scan (dicho valor se encontraba en SUMVAL+1).



Por último, sólo nos queda volver al juego 2 y cerrar el bucle de los scans. Ahora nos queda dibujar los atributos. Para ello, cargamos en HL la dirección destino de los atributos en la pantalla de memoria, en ATDIRG+1 el número de columnas que hay que rellenar , en SUMATT+1, 33-el número anterior (ya deberíais saber para qué lo vamos a utilizar), en E el byte de atributos, en D una máscara para las partes del atributo de la pantalla que no deben cambiar (el papel y el flash), y en A el número de filas a rellenar. En ATLAFX se cierra el bucle para cada fila, y en ATBLDI el de cada byte de la fila. Dentro de éste, tomamos el byte de atributos de la pantalla, nos quedamos con el PAPEL y el FLASH mediante un AND D, lo juntamos con INK y BRIGHT y lo guardamos en la pantalla. Por lo demás, es prácticamente idéntico este proceso al de volcar los atributos de pantalla a SPARES o de SPARES a pantalla, que ya ha sido visto. Tras dibujar los atributos, recuperamos A, que nos decía cuántos gráficos había que dibujar, y actualizamos IX para que apunte al siguiente elemento.

Con esto hemos terminado todo lo relacionado con el programa principal. Lo único que quedan son las tres subrutinas anexas, INICIO, DESACT y COMCHO, que son tan sencillas y poco interesantes y que no merecen una explicación a fondo.



SUGERENCIAS Y ADVERTENCIAS

Tal y como está, la zona de pantalla sobre la que se dibujan los sprites es toda la parte superior de ésta, las 22 líneas, pero no hay razón para que la zona usada tenga la altura que queramos. Por ejemplo, podemos reservar los dos tercios superiores para la pantalla de juego y utilizar el tercio inferior para marcadores y otras cosas. Cuanto más pequeña sea la zona destinada a sprites, mayor será la velocidad a la que vayan nuestros programas. Los ajustes que deberemos hacer son los siguientes: Si queremos que la zona en la que aparezcan los sprites tenga un alto de N filas, tendremos que hacer POKE 624ØØ,X, donde X es igual a N+1 si N está entre 1 y 8, a N+2 si N está entre 9 y 16, y a N+3 si N es mayor de 16. Si no queremos que esta zona empiece justo arriba de la pantalla podemos pokear en las direcciones 62387 y 62388 la dirección de la pantalla del ordenador dónde queremos que empiece, y en 62396 y 62397 la dirección de atributos.

Es necesario advertir que si usamos demasiados sprites o demasiado grandes, se puede producir el bloqueo del ordenador. La única solución a esto, como ya hemos dicho, es cambiar en el listado Ensamblador la ubicación de la zona SPARES.

Por último advertirles a los usuarios del Spectrum 128 o +2 que el programa funcionará perfectamente en modo 48 K, pero en 128 sólo funcionará si no intentamos interrumpir el programa con las interrupciones activadas, pues en este momento se bloqueará el ordenador.

Esperamos que este programa os sea de utilidad y consigáis con él hacer programas de calidad comercial.

LISTADO ENSAMBLADOR D

```
10 *D+
20 *C-
30 ;
40 ;    SPRITES  POR
50 ;
60 ;   INTERRUPCIONES
70 ;
80 ;  POR PABLO ARIZA.
90 ;
100 ;
110         ORG  62194
120 ;
130 ;
140 ENTINT DI
150         PUSH AF
160         PUSH BC
170         PUSH DE
180         PUSH HL
190         PUSH IX
200         EXX
210         EX   AF,AF'
220         PUSH AF
230         PUSH BC
240         PUSH DE
250         PUSH HL
260         PUSH IY
270         LD   A,(ESTADO)
280         AND  A
290         JP   NZ,NOVOLC
300         CALL VUELCA
310         LD   A,1
320         LD   (ESTADO),A
330         JP   SALINT
340 ;
350 NOVOLC LD    HL,SPARES
360         XOR  A
370         LD   (ESTADO),A
380         LD   A,(NUBLBO)
390         LD   IX,TABLDI
400 INBOTO DEC   A
410         JR   Z,INSABO
420         EX   AF,AF'
430         LD   E,(IX+0)
440         LD   D,(IX+1)
450         LD   A,(IX+14)
460         LD   (BORPOI+1),A
470         LD   (BORPOD+1),A
480         NEG
490         ADD  A,33
500         LD   (AUMPOI+1),A
510         LD   (AUMPOD+1),A
520         LD   A,(IX+6)
530         LD   B,0
540 BORPOI LD    C,0
550         LDIR
560 AUMPOI LD    C,0
570         EX   DE,HL
580         ADD  HL,BC
590         EX   DE,HL
600         DEC  A
610         JP   NZ,BORPOI
620         LD   A,(IX+13)
630         LD   E,(IX+11)
640         LD   D,(IX+12)
650 BORPOD LD    C,0
660         LDIR
670 AUMPOD LD    C,0
680         EX   DE,HL
690         ADD  HL,BC
700         EX   DE,HL
710         DEC  A
720         JP   NZ,BORPOD
730         LD   C,15
740         ADD  IX,BC
750         EX   AF,AF'
760         JP   INBOTO
770 INSABO LD    A,1
780         LD   (NUBLBO),A
790         LD   DE,SPARES
800         LD   IX,TABLDI
810         LD   IY,TASPRI
820         LD   A,(NUMSPR)
830         LD   B,A
840         AND  A
850         JR   Z,NOSPRI
860 INMAIN PUSH  BC
870         PUSH DE
880         CALL MOVERR
890         POP  DE
900         CALL CREADA
910         LD   BC,17
920         ADD  IY,BC
930         POP  BC
940         DJNZ INMAIN
950         CALL DIBUJA
960 NOSPRI POP   IY
970         PUSH IY
980         RST  56
990 SALINT POP   IY
1000        POP  HL
1010        POP  DE
1020        POP  BC
1030        POP  AF
1040        EX   AF,AF'
1050        EXX
1060        POP  IX
1070        POP  HL
1080        POP  DE
1090        POP  BC
1100        POP  AF
1110        EI
1120        RETI
1130 ;
1140 ;
1150 VUELCA LD   HL,ORIGSC
1160        LD   DE,16384
1170        LD   C,8
1180        EXX
1190        LD   HL,ORIGAT
1200        LD   DE,22528
1210        LD   BC,25*256
1220 LVOLCA EXX
1230        LD   B,9
1240 LOPPNU LDI
1250        LDI
1260        LDI
1270        LDI
1280        LDI
1290        LDI
1300        LDI
1310        LDI
1320        LDI
1330        LDI
1340        LDI
1350        LDI
1360        LDI
1370        LDI
1380        LDI
1390        LDI
1400        LDI
1410        LDI
1420        LDI
1430        LDI
1440        LDI
1450        LDI
1460        LDI
1470        LDI
1480        LDI
1490        LDI
1500        LDI
1510        LDI
1520        LDI
```

R DE MANEJO DE PANTALLAS

```
1530        LDI
1540        LDI
1550        LDI
1560        LD   A,E
1570        SUB  32
1580        LD   E,A
1590        JR   C,NOINCD
1600        INC  D
1610 NOINCD INC  HL
1620        DJNZ LOPPNU
1630        LD   A,E
1640        ADD  A,32
1650        LD   E,A
1660        JR   C,SITERC
1670        LD   A,D
1680        SUB  8
1690        LD   D,A
1700 SITERC EXX
1710        LDI
1720        LDI
1730        LDI
1740        LDI
1750        LDI
1760        LDI
1770        LDI
1780        LDI
1790        LDI
1800        LDI
1810        LDI
1820        LDI
1830        LDI
1840        LDI
1850        LDI
1860        LDI
1870        LDI
1880        LDI
1890        LDI
1900        LDI
1910        LDI
1920        LDI
1930        LDI
1940        LDI
1950        LDI
1960        LDI
1970        LDI
1980        LDI
1990        LDI
2000        LDI
2010        LDI
2020        LDI
2030        INC  HL
2040        DEC  B
2050        JP   NZ,LVOLCA
2060        RET
2070 ;
2080 ;
2090 MOVERR LD   A,(IY+5)
2100        CP   127
2110        RET  Z
2120        LD   A,(IY+9)
2130        LD   D,A
2140        AND  71
2150        LD   (IX+10),A
2160        BIT  4,D
2170        JR   Z,RECLIN
2180        LD   L,(IY+13)
2190        LD   H,(IY+14)
2200 RECUPE LD   A,(HL)
2210        CP   127
2220        JR   NZ,NOFITA
2230        LD   L,(IY+10)
2240        LD   H,(IY+11)
2250        LD   A,(HL)
2260 NOFITA CP   126
2270        JR   NZ,TRANOR
2280        INC  HL
2290        LD   A,(HL)
2300        LD   (IY+0),A
2310        INC  HL
2320        LD   A,(HL)
2330        LD   (IY+1),A
2340        INC  HL
2350        JP   RECUPE
2360 TRANOR LD   E,A
2370        INC  HL
2380        LD   A,(HL)
2390        INC  HL
2400        LD   (IY+13),L
2410        LD   (IY+14),H
2420        JP   COCORE
2430 RECLIN LD   A,(IY+10)
2440        LD   E,A
2450        LD   A,(IY+11)
2460 COCORE LD   C,A
2470        CP   128
2480        CCF
2490        SBC  A,A
2500        LD   B,A
2510        LD   L,(IY+6)
2520        LD   H,(IY+7)
2530        ADD  HL,BC
2540        LD   (IY+6),L
2550        LD   (IY+7),H
2560        LD   A,E
2570        LD   C,A
2580        CP   128
2590        CCF
2600        SBC  A,A
2610        LD   B,A
2620        LD   L,(IY+4)
2630        LD   H,(IY+5)
2640        ADD  HL,BC
2650        LD   (IY+4),L
2660        LD   (IY+5),H
2670        LD   A,(IY+12)
2680        LD   E,(IY+8)
2690        BIT  3,D
2700        JP   NZ,ADETRA
2710        INC  A
2720        CP   E
2730        JR   NZ,NOFICI
2740        XOR  A
2750 NOFICI LD   (IY+12),A
2760        JP   TRAREN
2770 ADETRA INC  A
2780        LD   D,A
2790        SUB  E
2800        SUB  E
2810        ADD  A,2
2820        LD   (PARETR+1),A
2830        JR   Z,FIANCI
2840        LD   A,D
2850 FIANCI LD   (IY+12),A
2860        CP   E
2870        JR   C,TRAREN
2880 PARETR LD   A,0
2890        NEG
2900 TRAREN LD   E,(IY+15)
2910        LD   D,(IY+16)
2920        SLA  E
2930        RL   D
2940        LD   L,(IY+0)
2950        LD   H,(IY+1)
2960        AND  A
2970        JR   Z,PRIFAS
2980        LD   B,A
2990 MULTIP ADD  HL,DE
3000        DJNZ MULTIP
3010 PRIFAS LD   (IX+2),L
3020        LD   (IX+3),H
3030        RET
3040 ;
3050 ;
3060 CREADA LD   A,(IY+4)
3070        LD   C,A
3080        AND  6
3090        ADD  A,248
3100        LD   (IX+9),A
3110        LD   A,(IY+5)
3120        AND  A
3130        JR   Z,XPOSIT
3140        INC  A
3150        RET  NZ
3160        LD   A,C
3170        ADD  A,6
3180        JR   NC,NEGARE
3190        LD   HL,ORIGAT-1
3200        LD   BC,ORIGSC-1
3210        LD   A,(IY+2)
3220        LD   (IX+7),A
3230        XOR  A
3240        LD   (IX+8),A
3250        JP   PARTEY
3260 NEGARE SCF
3270        RRA
3280        SRA  A
3290        SRA  A
3300        LD   B,A
3310        ADD  A,(IY+2)
3320        RET  NC
3330        RET  Z
3340        LD   (IX+7),A
3350        LD   A,B
3360        NEG
3370        LD   (IX+8),A
3380        ADD  A,(IX+2)
3390        LD   (IX+2),A
3400        LD   A,0
3410        ADC  A,(IX+3)
3420        LD   (IX+3),A
3430        LD   A,C
3440        LD   HL,ORIGAT-1
3450        LD   BC,ORIGSC-1
3460        AND  6
3470        JR   NZ,PARTEY
3480        INC  HL
3490        INC  BC
3500        JP   PARTEY
3510 XPOSIT LD   A,C
3520        SUB  1
3530        JR   C,INTEGE
3540        SRL  A
3550        SRL  A
3560        SRL  A
3570        ADD  A,(IY+2)
3580        CP   33
3590        JR   C,INTEGE
3600        SUB  32
3610        LD   (IX+8),A
3620        NEG
3630        ADD  A,(IY+2)
3640        LD   (IX+7),A
3650 RECOIN LD   A,C
3660        SRL  A
3670        SRL  A
3680        SRL  A
3690        LD   C,A
3700        LD   HL,ORIGAT
3710        ADD  A,L
3720        LD   L,A
3730        ADC  A,H
3740        SUB  L
3750        LD   H,A
3760        LD   A,C
3770        LD   BC,ORIGSC
3780        ADD  A,C
3790        LD   C,A
3800        ADC  A,B
3810        SUB  C
3820        LD   B,A
3830        JP   PARTEY
3840 INTEGE LD   A,(IY+2)
3850        LD   (IX+7),A
3860        XOR  A
3870        LD   (IX+8),A
3880        JP   RECOIN
3890 PARTEY LD   A,(IY+7)
3900        AND  A
3910        JR   Z,YPOSIT
3920        INC  A
3930        RET  NZ
3940        LD   A,(IY+6)
3950        ADD  A,(IY+3)
3960        RET  NC
3970        RET  Z
3980        LD   (IX+6),A
3990        NEG
4000        ADD  A,(IY+3)
4010        PUSH HL
4020        PUSH DE
4030        LD   L,(IX+2)
4040        LD   H,(IX+3)
4050        LD   D,0
4060        LD   E,(IY+2)
4070 NESUDI ADD  HL,DE
4080        DEC  A
4090        JP   NZ,NESUDI
4100        LD   (IX+2),L
4110        LD   (IX+3),H
4120        LD   A,(IX+6)
4130        ADD  A,7
4140        SRL  A
4150        SRL  A
4160        SRL  A
4170        LD   (IX+13),A
4180        JP   COYPOS
4190 YPOSIT LD   A,(IY+6)
4200        CP   192
4210        RET  NC
4220        SRL  A
4230        SRL  A
4240        SRL  A
4250        PUSH BC
4260        LD   BC,33
4270        INC  A
4280 CALATT DEC  A
4290        JR   Z,ATCOMP
4300        ADD  HL,BC
4310        JP   CALATT
4320 ATCOMP POP  BC
4330        PUSH HL
4340        PUSH DE
4350        LD   H,B
4360        LD   L,C
4370        LD   DE,33
4380        LD   A,(IY+6)
4390        AND  A
4400        JR   Z,PRIFIL
4410 CALFIL ADD  HL,DE
4420        DEC  A
4430        JP   NZ,CALFIL
4440        LD   B,H
4450        LD   C,L
4460 PRIFIL LD   A,(IY+6)
4470        ADD  A,(IY+3)
4480        JR   C,MOSCAB
4490        CP   193
4500        JR   NC,MOSCAB
4510        LD   A,(IY+3)
4520        LD   (IX+6),A
4530        LD   A,C
4540        EX   AF,AF'
4550        LD   C,(IY+6)
4560        SRL  C
4570        SRL  C
4580        SRL  C
4590        LD   A,(IY+6)
4600        ADD  A,(IY+3)
4610        ADD  A,7
4620        SRL  A
4630        SRL  A
4640        SRL  A
4650        SUB  C
4660        LD   (IX+13),A
4670        EX   AF,AF'
4680        LD   C,A
4690        JP   COYPOS
4700 MOSCAB LD   A,192
4710        SUB  (IY+6)
4720        LD   (IX+6),A
4730        ADD  A,7
4740        SRL  A
4750        SRL  A
4760        SRL  A
4770        LD   (IX+13),A
4780 COYPOS LD   (IX+0),C
4790        LD   (IX+1),B
4800        LD   L,(IX+2)
4810        LD   H,(IX+3)
4820        LD   E,(IY+15)
4830        LD   D,(IY+16)
4840        ADD  HL,DE
4850        LD   (IX+4),L
4860        LD   (IX+5),H
4870        LD   D,(IX+7)
4880        LD   A,(IX+9)
4890        AND  6
4900        JR   Z,SANUBY
4910        INC  D
4920 SANUBY LD   (IX+14),D
4930        POP  DE
4940        POP  HL
4950        PUSH HL
4960        LD   (IX+11),L
4970        LD   (IX+12),H
4980        LD   H,B
4990        LD   L,C
5000        LD   A,(IX+14)
5010        LD   (PUBYSA+1),A
5020        LD   (PUATSA+1),A
5030        NEG
5040        ADD  A,33
5050        LD   (PUSUSA+1),A
5060        LD   (ATSUSA+1),A
5070        LD   A,(NUBLBO)
5080        INC  A
5090        LD   (NUBLBO),A
5100        LD   A,(IX+6)
5110        LD   B,0
5120 PUBYSA LD   C,0
5130        LDIR
5140 PUSUSA LD   C,0
5150        ADD  HL,BC
5160        DEC  A
5170        JR   NZ,PUBYSA
5180        POP  HL
5190        LD   A,(IX+13)
5200 PUATSA LD   C,0
5210        LDIR
5220 ATSUSA LD   C,0
5230        ADD  HL,BC
5240        DEC  A
5250        JP   NZ,PUATSA
5260        LD   C,15
5270        ADD  IX,BC
5280        RET
5290 ;
5300 ;
5310 DIBUJA LD   IX,TABLDI
5320        LD   A,(NUBLBO)
```

```
5330 DIBLUP DEC  A
5340        RET  Z
5350        PUSH AF
5360        LD   E,(IX+0)
5370        LD   D,(IX+1)
5380        LD   H,(IX+9)
5390        LD   L,255
5400        LD   A,(HL)
5410        LD   (PONUDO+1),A
5420        INC  H
5430        LD   A,(HL)
5440        DEC  H
5450        LD   (POSNUM+1),A
5460        EXX
5470        LD   L,(IX+2)
5480        LD   H,(IX+3)
5490        PUSH HL
5500        POP  IY
5510        LD   L,(IX+4)
5520        LD   H,(IX+5)
5530        LD   A,(IX+7)
5540        LD   (POSBYT+1),A
5550        NEG
5560        ADD  A,33
5570        LD   (SUMVAL+1),A
5580        LD   C,(IX+6)
5590        LD   E,(IX+8)
5600        LD   D,0
5610 LOSCAN EXX
5620 POSNUM LD   BC,0
5630        EXX
5640 POSBYT LD   B,0
5650 LOBYTS LD   A,(IY+0)
5660        INC  IY
5670        EX   AF,AF'
5680        LD   A,(HL)
5690        INC  HL
5700        EXX
5710        LD   L,A
5720        LD   A,(HL)
5730        OR   C
5740        INC  H
5750        LD   C,(HL)
5760        EX   AF,AF'
5770        LD   L,A
5780        LD   A,B
5790        LD   B,(HL)
5800        DEC  H
5810        OR   (HL)
5820        LD   L,A
5830        EX   AF,AF'
5840        EX   DE,HL
5850        AND  (HL)
5860        OR   E
5870        LD   (HL),A
5880        INC  HL
5890        EX   DE,HL
5900        EXX
5910        DJNZ LOBYTS
5920        ADD  HL,DE
5930        ADD  IY,DE
5940        EXX
5950        LD   A,C
5960 PONUDO OR   0
5970        EX   DE,HL
5980        AND  (HL)
5990        OR   B
6000        LD   (HL),A
6010 SUMVAL LD   A,0
6020        ADD  A,L
6030        LD   L,A
6040        ADC  A,H
6050        SUB  L
6060        LD   H,A
6070        EX   DE,HL
6080        EXX
6090        DEC  C
6100        JP   NZ,LOSCAN
6110        LD   L,(IX+11)
6120        LD   H,(IX+12)
6130        LD   A,(IX+14)
6140        LD   (ATDIRG+1),A
6150        NEG
6160        ADD  A,33
6170        LD   (SUMATT+1),A
6180        LD   E,(IX+10)
6190        LD   D,184
6200        LD   A,(IX+13)
6210 ATLAFX EX   AF,AF'
6220 ATDIRG LD   B,0
6230 ATBLDI LD   A,(HL)
6240        AND  D
6250        OR   E
6260        LD   (HL),A
6270        INC  HL
6280        DJNZ ATBLDI
6290 SUMATT LD   C,0
6300        ADD  HL,BC
6310        EX   AF,AF'
6320        DEC  A
6330        JP   NZ,ATLAFX
6340        POP  AF
6350        LD   C,15
6360        ADD  IX,BC
6370        JP   DIBLUP
6380 ;
6390 ;
6400 ESTADO DEFB 0
6410 NUBLBO DEFB 1
6420 ;
6430 ;
6440 DIMEXS LD   E,(IY+2)
6450        SLA  E
6460        SLA  E
6470        SLA  E
6480        JP   CONTIN
6490 DIMEYS LD   E,(IY+3)
6500 CONTIN LD   D,0
6510        AND  A
6520        SBC  HL,DE
6530        RET
6540 COORDS CALL DITABS
6550 CALPOI JP   0
6560 COORDX LD   L,(IY+4)
6570        LD   H,(IY+5)
6580        RET
6590 COORDY LD   L,(IY+6)
6600        LD   H,(IY+7)
6610        RET
6620 DITABS LD   IY,TASPRI
6630        AND  A
6640        RET  Z
6650        PUSH DE
6660        LD   DE,17
6670 SUMDIS ADD  IY,DE
6680        DEC  A
6690        JR   NZ,SUMDIS
6700        POP  DE
6710        RET
6720 ;
6730 ;
6740 NUMSPR EQU  23681
6750 TABLDI EQU  53289
6760 TASPRI EQU  53000
6770 ORIGSC EQU  54568
6780 ORIGAT EQU  60904
6790 SPARES EQU  53544
6800 SPRICH EQU  23728
6810 ;
6820 ;
6830        ORG  61953
6840 INICIO DI
6850        LD   A,241
6860        LD   I,A
6870        LD   B,0
6880        LD   HL,61696
6890 INTCRE LD   (HL),242
6900        INC  HL
6910        DJNZ INTCRE
6920        LD   (HL),242
6930        LD   DE,ORIGSC
6940        LD   HL,16384
6950        LD   A,192
6960 INILAP EX   AF,AF'
6970        PUSH HL
6980        LD   BC,32
6990        LDIR
7000        INC  DE
7010        POP  HL
7020        INC  H
7030        LD   A,H
7040        AND  7
7050        JR   NZ,INIPRT
7060        LD   A,L
7070        ADD  A,32
7080        LD   L,A
7090        JR   C,INIPRT
7100        LD   A,H
7110        SUB  8
7120        LD   H,A
7130 INIPRT EX   AF,AF'
7140        DEC  A
7150        JR   NZ,INILAP
7160        LD   HL,22528
7170        LD   DE,ORIGAT
7180        LD   A,24
7190 INATLA LD   BC,32
7200        LDIR
7210        INC  DE
7220        DEC  A
7230        JR   NZ,INATLA
7240        IM   2
7250        LD   C,4
7260        LD   H,248
7270 LCRTCR LD   A,9
7280        SUB  C
7290        SUB  C
7300        LD   L,0
7310 ROTACU LD   B,L
7320        LD   E,0
7330        LD   D,A
7340 VERROT DEC  D
7350        JR   Z,VEREXI
7360        SRL  B
7370        RR   E
7380        JR   VERROT
7390 VEREXI LD   (HL),B
7400        INC  H
7410        LD   (HL),E
7420        DEC  H
7430        DEC  L
7440        JR   NZ,ROTACU
7450        INC  H
7460        INC  H
7470        DEC  C
7480        JR   NZ,LCRTCR
7490        LD   A,1
7500        LD   (NUBLBO),A
7510        XOR  A
7520        LD   (NUMSPR),A
7530        LD   (ESTADO),A
7540        EI
7550        RET
7560 ;
7570 ;
7580 DESACT LD   A,63
7590        IM   1
7600        LD   I,A
7610        RET
7620 ;
7630 ;
7640 COMCHO LD   BC,(SPRICH)
7650        INC  C
7660        JR   Z,BCONTO
7670        DEC  C
7680        CALL CHOSUB
7690        LD   BC,0
7700        RET  NC
7710        INC  BC
7720        RET
7730 ;
7740 BCONTO LD   IX,SPRICH
7750        LD   A,(NUMSPR)
7760 BUCHAL INC  (IX+0)
7770        EX   AF,AF'
7780        LD   BC,(SPRICH)
7790        LD   A,B
7800        CP   C
7810        JR   Z,SASPRI
7820        CALL CHOSUB
7830        LD   BC,1
7840        RET  C
7850 SASPRI EX   AF,AF'
7860        DEC  A
7870        JR   NZ,BUCHAL
7880        DEC  BC
7890        RET
7900 ;
7910 CHOSUB LD   HL,COORDX
7920        LD   (CALPOI+1),HL
7930        LD   HL,DIMEXS
7940        LD   (ANCALT+1),HL
7950        CALL CHOREA
7960        RET  NC
7970        LD   HL,COORDY
7980        LD   (CALPOI+1),HL
7990        LD   HL,DIMEYS
8000        LD   (ANCALT+1),HL
8010 CHOREA LD   A,B
8020        CALL COORDS
8030        EX   DE,HL
8040        LD   A,C
8050        CALL COORDS
8060        AND  A
8070        SBC  HL,DE
8080        LD   A,B
8090        JR   NC,BIENCO
8100        LD   A,H
8110        CPL
8120        LD   H,A
8130        LD   A,L
8140        CPL
8150        LD   L,A
8160        INC  HL
8170        LD   A,C
8180 BIENCO CALL DITABS
8190 ANCALT JP   0
```