

BRÜCKMANN SCHIEB

EL GRAN LIBRO DEL

FLOPPY CPC^{664/6128}

**PROGRAMACION DEL DISKETTE PARA
PRINCIPIANTES, AVANZADOS Y
PROFESIONALES**

**UN LIBRO DATA BECKER
EDITADO POR FERRE MORET, S.A.**

BRÜCKMANN SCHIEB

EL GRAN LIBRO DEL

FLOPPY CPC 664 / 6128

**PROGRAMACION DEL DISKETTE PARA
PRINCIPIANTES, AVANZADOS Y
PROFESIONALES**

**UN LIBRO DATA BECKER
EDITADO POR FERRE MORET, S.A.**

Este libro ha sido traducido por Joaquin Ramis.

Imprime: APSSA, ROCA UMBERT, 26 - L'HOSPITALET DE LL. (Barcelona)

ISBN 84-86437-58-X

Depósito legal B- 26788/86

**Copyright (C) 1985 DATA BECKER GmbH
Merowingerstr.30
4000 Düsseldorf**

**Copyright (C) 1986 FERRE MORET,S.A.
Córsega, 299
08008 BARCELONA**

**Reservados todos los derechos. Ninguna parte de este libro
podrá ser reproducida de algún modo (impresión, fotocopia o
quiero otro procedimiento) o bien, utilizado, reproducido
o difundido mediante sistemas electrónicos sin la
autorización previa de FERRE MORET, S.A.**

Advertencia importante

Los circuitos, procedimientos y programas reproducidos en este libro son divulgados sin tener en cuenta el estado de las patentes. Están destinados exclusivamente al uso amateur o docente, y no pueden ser utilizados para fines comerciales. Todos los circuitos, datos técnicos y programas de este libro, han sido elaborados o recopilados con el mayor cuidado por el autor y reproducidos utilizando medidas de control eficaces. No obstante, es posible que exista algún error. FERRE MORET, S.A. se ve por tanto obligada a advertirles, que no puede asumir ninguna garantía, ni responsabilidad jurídica, ni cualquier otra responsabilidad sobre las consecuencias atribuibles a datos erróneos. El autor les agradecerá en todo momento la comunicación de posibles fallos.

P R O L O G O

El presente Gran Libro del Floppy para el CPC trata en esta segunda versiòn de las unidades de disco del CPC 464, del CPC 664 y del CPC 6128 conjuntamente. Se comentan fallos del sistema operativo y se solventan insuficiencias de la unidad de discos o, en su caso, del AMSDOS (mensajes de error en pantalla; gestiòn de ficheros relativos).

La abundante informaciòn sobre almacenamiento secuencial de datos viene acompañada de mÙltiples programas a tÙtulo de ejemplo y de ilustraciones para el principiante. El programador interesado en aspectos mÙs tÙcnicos hallarà en este libro, ademàs, lo mÙs digno de conocer de "su" floppy (programaciòn del controlador, etc.).

Los autores fueron considerablemente auxiliados por los lectores en la confecciòn de la presente obra; la lectora Brigitte Witzer en especial destaca en ello, por lo que deseamos expresarle nuestro agradecimiento.

Por fin le deseamos, querido lector, que pase ratos agradables con este libro, y mucho éxito en las tareas con su ordenador Amstrad y su unidad de disco.

Düsseldorf, Febrero de 1986

Rolf Brückmann
Jörg Schieb

I N D I C E

CAPITULO 1 Introducción a la programación del DDI-1

1.1	¿Qué ofrece un floppy?	13
1.2	El diskette	16
1.2.1	El desarrollo	16
1.2.2	El disco incluido	20
1.3	Algo sobre CP/M	21
1.3.1	¿Qué es CP/M?	21
1.3.2	Carga y puesta en marcha del CP/M	22
1.3.3	Formateado de discos	24
1.3.4	Copiar en CP/M	27
1.3.5	Copiar con dos unidades de disco	29
1.3.6	DISCCHK - Comprobación de discos	30
1.3.7	Formatos especiales de formateado	31
1.3.8	Ordenes en tránsito y residentes	34
1.3.9	Copia de ficheros mediante FILECOPY	35
1.3.10	Wild cards	38
1.3.11	El comando DIR	39
1.3.12	El comando ERA	40
1.3.13	El comando REN	41
1.3.14	El comando TYPE	41
1.3.15	Cambio de la unidad estàndar de disco	41
1.3.16	El comando PIP	42
1.3.17	El comando STAT	43
1.3.18	BOOTGEN	45
1.3.19	MOVCPM y SYSGEN	45
1.3.20	SETUP	46
1.3.21	AMSDOS	46
1.4	Trabajo con AMSDOS	47
1.4.1	¿Qué es AMSDOS?	47
1.4.2	LOAD y RUN	49
1.4.3	Las órdenes de fichero	54
1.4.4	Las órdenes adicionales del AMSDOS	62

1.5	Almacenamiento secuencial de datos.....	74
1.5.1	¿Qué es almacenamiento secuencial de datos?	74
1.5.2	Programación de ficheros secuenciales	79
1.5.3	Ficheros secuenciales y variables de campo	87
1.5.4	Diferencias entre PRINT# y WRITE#	98
1.5.5	Diferencias entre INPUT# y LINE INPUT#	103
1.6	Almacenamiento relativo de datos	108
1.6.1	¿Qué es almacenamiento relativo de datos?	108
1.6.2	La creación de un fichero relativo	111
1.6.3	¿Cómo trabaja con almacenamiento relativo?	114
1.6.4	?Cómo funciona el almacenamiento relativo?	116
1.6.5	¿Cuándo es lógico el almacenamiento relativo? ...	118
1.6.6	Los ficheros ISAM	119
1.7	Los distintos ordenadores Amstrad	121
1.7.1	El CPC 664 y el CPC 6128 y errores de disco	122
1.7.2	Otras diferencias entre el CPC 664 y el CPC 6128	123

CAPITULO 2

Programación con discos para avanzados

2.1	Los vectores del DOS	125
2.1.1	DISC IN OPEN	128
2.1.2	DISC IN CLOSE	131
2.1.3	DISC IN ABANDON	132
2.1.4	DISC IN CHAR	132
2.1.5	DISC IN DIR	133
2.1.6	DISC RETURN	134
2.1.7	DISC TEST EOF	135
2.1.8	DISC OUT OPEN	136
2.1.9	DISC OUT CLOSE	139
2.1.10	DISC OUT ABANDON	140
2.1.11	DISC OUT CHAR	140
2.1.12	DISC OUT DIRECT	141
2.1.13	DISC CATALOG	142
2.1.14	Remendando vectores	143

2.2	Las ampliaciones de órdenes del AMSDOS	145
2.2.1	Programaciòn en ensamblador de las ampliaciones	146
2.2.2	Las órdenes 'ocultas' de la ampliaciòn	151
2.2.2.1	La orden &81 Message on/off	151
2.2.2.2	La orden &82 Drive Parameter	152
2.2.2.3	La orden &83 Disc Format Parameter	155
2.2.2.4	La orden &84 Read Sector	156
2.2.2.5	La orden &85 Write Sector	160
2.2.2.6	La orden &86 Format Track	160
2.2.2.7	La orden &87 Seek Track	163
2.2.2.8	La orden &88 Test Drive	164
2.2.2.9	La orden &89 Retry Count	164

CAPITULO 3 Técnica del floppy y del disco

3.1	El controlador del floppy	167
3.1.1	Descripción del hardware del floppy controller .	169
3.1.2	El FDC 765	170
3.1.2.1	Disposición de conexiones del FDC	172
3.1.2.2	Programaciòn del FDC 765	180
3.1.2.3	El registro de estado del FDC 765	198
3.1.2.4	Inserciòn del FDC 765 en el CPC	207
3.2	Estructura del disco	209
3.2.1	Los tres formatos de disco	209
3.2.1.1	El formato estàndar CPC o formato de sistema....	210
3.2.1.2	El formato de datos	211
3.2.1.3	El formato IBM	211
3.2.2	Estructura del directorio	212
3.2.3	Estructura de los ficheros	217
3.2.4	El registro fisioco de los datos	220
3.2.4.1	MF, MFM, bits y magnetismo	220
3.2.4.2	Sobre GAPs, IDs y Adress-Marks	228

CAPITULO 4**La ROM y la RAM del AMSDOS**

4.1	La RAM de sistema del AMSDOS	235
4.2	La ROM del AMSDOS	247

CAPITULO 5**Programas y trucos para el DDI-1**

5.1	Errores en MERGE y CHAIN MERGE	329
5.2	Mensajes de error	334
5.2.1	Explicaciòn de la rutina de intercepcione de errores - ¿Còmo se usa?	346
5.3	Gestione de ficheros relativos	350
5.3.1	Explicaciòn del programa en lenguaje mquina ..	360
5.4	Programa de gestiòn de ficheros relativos	378
5.5	Monitor de disco	395
5.6	Manager de discos	403
	Indice alfabetico	413

CAPITULO 1**Introducción a la programación del DDI-1**

Los ordenadores CPC son aparatos apreciados y que se caracterizan por su buena relación calidad precio. Como ya sabe, existen en el mercado tres ordenadores CPC distintos. El libro del floppy ha de ocuparse de estos tres aparatos: el CPC 464, el CPC 664 y el CPC 6128.

Si bien el aspecto exterior de los tres ordenadores es diferente (distinto teclado etc.), el servicio y mantenimiento de sus unidades de disco es (casi) totalmente idéntico. Sin embargo, y debido a las distintas versiones de BASIC se hace a veces necesario efectuar las oportunas distinciones. En caso de que no se indique lo contrario, lo dicho vale por igual para los tres ordenadores CPC.

1.1 ¿Qué ofrece un floppy?

Todos los ordenadores CPC (como cualquier otro ordenador) pueden disponer tan sólo de una memoria temporal - debido a ello, los datos y programas introducidos en el ordenador se pierden al desconectar dicho ordenador. Para almacenar estas informaciones, para conservarlas, prácticamente, precisamos de memorias externas, como, por ejemplo, una unidad de cassettes o bien una unidad de disco.

El CPC 464 dispone de una unidad de cassettes - la unidad de disco es por tanto únicamente una memoria externa opcional (es decir, adicional). El CPC 664 y el CPC 6128 ya llevan la unidad de disco incorporada en la carcasa de sus teclados en la versión estándar. Las unidades de disco incorporadas no se diferencian en absoluto de las opcionales.

Si posee usted un CPC 464, habrá descubierto ya en su manual que puede escribir en cassette a dos velocidades distintas. En la lectura, el ordenador se ajustará automáticamente a la

velocidad de lectura correspondiente. Mediante la orden SPEED WRITE puede usted obligar a la unidad de cassette la grabaciòn a 1000 baudios o a 2000 baudios. ¿Y què significa baudio? Un baudio es igual a un bit por segundo. Por tanto, 1000 baudios significa que el ordenador escribe en el cassette, y mès tarde lee 1000 bits por segundo. 1000 bits son $1000/8=125$ bytes; correspondientemente, con 2000 baudios se leen o escriben 250 bytes por segundo. ¿Y porquè no se trabaja siempre con 2000 baudios?, se preguntarà usted. Muy sencillo: Con 2000 baudios, el peligro de un error en escritura/lectura es mucho mayor, es decir, que en la lectura de un programa no se lea lo mismo que se escribió. Imagine lo que ello representaria.

Con un floppy tenemos la ventaja de que, tras un error de lectura se puede iniciar un nuevo intento de lectura (en una unidad de cassettes tendríamos que rebobinar manualmente o mediante una complicada mecànica).

Normalmente se efectúan hasta diez intentos de lectura de un sector antes de que se envíe el mensaje de que el sector no es legible. En la unidad de cassettes, dicho mensaje ha de darse directamente después de la detección del error.

En realidad, una unidad de cassette no està indicada en absoluto para el uso profesional. Los dos motivos fundamentales son:

1. Una unidad de cassette es demasiado lenta. Los programas profesionales son cada vez mès amplios, y, por tanto, mès largos, no es raro que ocupen 25 KBytes, lo que conlleva largos periodos de carga. Además, después aún han de almacenar y leerse datos con relativa frecuencia con lo que nos hallariamos con el segundo problema:
2. Es muy dificil cargar ficheros concretos de cassette. Un fichero representa, por ejemplo, un programa almacenado o datos almacenados. Si tiene varios programas cargados en una misma cinta, o bien ha de avanzar la cinta con

dificultad y sin exactitud y tratar después de cargar, o bien deja al ordenador la búsqueda del fichero - lo que significa que leerá todos los demás ficheros.

Estas desventajas de la utilización de cassettes se pueden reducir: los diskettes ofrecen un trabajo más rápido y un manejo más sencillo; son los medios de almacenamiento de forma laminar que se introducen en una unidad de disco.

En grandes instalaciones se utilizan los discos duros, los hermanos mayores de los diskettes. Un disco duro puede tener una capacidad de hasta 50 MBytes (50 Mega = 50 millones). Pero aún hoy se copian datos importantes en las lentes cintas magnéticas, sobre todo cuando se desea archivar datos. Las cintas magnéticas son muy indicadas para ello porque ocupan poco espacio y son más baratas.

Pero volviendo al CPC y la utilización de diskettes:

"Bienvenidos al círculo de los orgullosos usuarios de Amstrad DDI-1. Pronto verá lo acertado de su decisión y que su inversión ha valido la pena..."

Esto es lo que le promete el manual de su unidad de disco Amstrad DDI-1. Si posee un CPC 646 o bien un CPC 6128, la promesa se hace también extensiva a usted. Mediante este libro queremos ayudarle a hacer realidad esta introducción, mostrándole las posibilidades ocultas en este floppy.

1.2 El diskette

1.2.1 El desarrollo

Hasta hace poco, la mayoría de los diskettes median ocho pulgadas. Pero estos discos se mostraron poco prácticos; algo más tarde fueron desarrollados los discos de 5 1/4 pulgadas, tal y como se utilizan actualmente en casi todos los ordenadores personales y domésticos usuales (IBM, Apple, Commodore 64, Sirius, etc.). Se produjo una revolución con el LISA de Apple y con el más reducido Macintosh, que utiliza discos de 3 1/2 pulgadas - el FDD de Amstrad trabaja incluso con discos de tres pulgadas. Así pues, la miniaturización avanza notablemente incluso en el campo de la memoria externa.

Dado que aún se producen muy pocos discos de tres pulgadas, éstos son, hasta ahora, relativamente caros. Pero no habría de hacerse esperar una caída de precios, de modo que dentro de un año deberían costar aproximadamente la mitad.

Seguro que conoce los discos de 5 1/4 pulgadas, que constan de una funda protectora de plástico flexible y de una placa interior a ella. En el caso de sus discos, estas placas están protegidas por motivos de seguridad mediante una carcasa estable de plástico. De esta manera ya no puede tocar con el dedo la abertura para el cabezal de escritura/lectura (figura 1, punto 1). Esta abertura está provista de un cierre metálico que no se abre en tanto el disco no sea introducido en la unidad.

Lo mismo vale para el agujero de alineación (punto 2). Los discos tienen un agujero de alineación para que la unidad de disco "sepa" siempre cuando se repite una vuelta completa. Dichas perforaciones son reconocidas mediante un sensor óptico. En la práctica, sirven para la orientación. También existen unidades de disco que no las precisan, pero en cualquier caso son más lentas, ya que en tal caso, dicho problema ha de ser solucionado por software.

Ambas aberturas están aseguradas mediante un cierre plegable, que puede usted abrir con cuidado por una vez. Ponga el disco en su mano, con la cara B hacia arriba y la abertura de escritura/lectura desviada de usted. Siga la guía izquierda con la uña hasta que reconozca una corredera blanca de plástico (3). Tire de ella hacia usted hasta el tope. Observará que tanto la perforación de índice como la abertura para el cabezal de escritura/lectura quedan liberadas. Ahora ve directamente el disco en si, en el que están almacenadas las informaciones. Pero no lo toque con los dedos. En el centro (4) se halla el agujero del eje motriz. Aquí es engarzado y girado el disco. Si con la otra mano gira cuidadosamente la placa, verá también el agujero de alineación. Vuelva a cerrar por fin cuidadosamente el diskette soltando la corredera blanca de plástico.

Finalmente cabría mencionar las dos perforaciones que sirven para protección de la escritura (5). En el borde izquierdo de cada disco se halla una perforación provista de una pequeña flecha. Sirve para proteger un disco de una eventual sobreescritura. Esto ya lo conoce de su cassette. Cuando la perforación esté cerrada, puede escribir en el disco. Pero también puede deslizar la corredera de modo que el agujero quede abierto. En tal caso no se puede escribir en el disco, lo que tiene pleno sentido si en él tenemos valiosos programas que no deben ser borrados involuntariamente. En los discos de Amstrad que acompañan al aparato, en los que se hallan CP/M y LOGO, ha de deslizar la corredera de arriba hacia abajo. En discos de otros fabricantes esto funciona de un modo algo diferente, pero igualmente sencillo. En los discos que nosotros utilizamos, usted ha de mover la corredera de derecha a izquierda p.ej. con un bolígrafo.

Sus discos pueden ser utilizados por ambas caras. En los discos de 5 1/4 pulgadas se explicitaba esta versión con el título "Double Sided"; un lujo que había que pagar en consecuencia. En su Amstrad se sobreentiende que puede utilizar ambas caras de su diskette.

En cada cara caben 180 KBytes, de modo que en un disco se pueden almacenar 360 KBytes o bien 360.000 caracteres en total.

Para introducir un disco en la unidad simplemente ha de tomar el disco en su mano e introducirlo cuidadosamente en el sentido de la flecha, hasta que oiga el sonido de chasquido: el disco estará entonces firmemente encajado en la unidad. Para volver a sacar el disco de la unidad, oprima el pulsador que se halla a la derecha de la ranura de inserción. A este pulsador se le llama también pulsador eyector de disco. ¡No lo pulse jamás mientras su unidad de disco esté trabajando! Ello puede conducir a una pérdida total de los datos.

Preste también atención, tanto al conectar como al desconectar la unidad, a que en ella no se halle diskette alguno — también en ese caso puede producirse una pérdida de datos debido a picos de tensión en el cabezal de escritura/lectura.

Un disco nuevo todavía está "crudo", es, en cierto modo, una hoja en blanco. Antes de poder utilizar un disco, éste ha de ser formateado, similarmente a una hoja en blanco, que ha de ser provista de unas líneas para una correcta escritura. El modo de formatear de nuevo un disco, le será explicado en el capítulo 1.3.3 (CP/M).

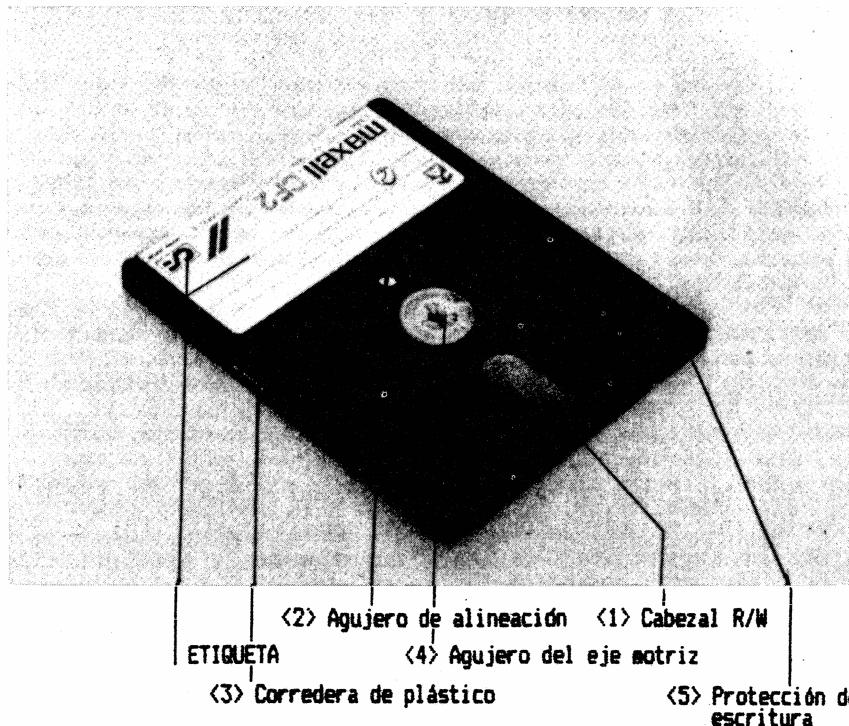


Figura 1. Disco de 3 pulgadas

1.2.2 El disco incluido en el equipo

Al comprar la DDI-1 de Amstrad, y por supuesto también al comprar el CPC 664 o el CPC 6128, usted recibió un disco de sistema, o bien dos discos de sistema, en el caso del CPC 6128. En este disco de sistema se halla el sistema operativo CP/M, y en la cara B se incluye, además, Dr.LOGO. El lenguaje LOGO ha obtenido una gran aceptación y fue creado para posibilitar la ''programación'' de ordenadores incluso a niños. Si desea usar el LOGO, hallará literatura especializada al respecto.

En el manual adjunto al disco se hallan todas las órdenes LOGO aclaradas con ejemplos relativamente cortos. Dr. LOGO significa Digital Research LOGO y es una versión del LOGO adaptada al Amstrad CPC. El lenguaje de programación LOGO ha sido ampliado en algunas órdenes de sonido, para aprovechar las posibilidades sonoras del Amstrad CPC. Además, se incluyeron las teclas cursoras para la edición del programa.

1.3. Algo sobre CP/M

1.3.1 ¿Qué es CP/M?

En la cara A de su disco se halla el sistema operativo CP/M. CP/M significa Control Program for Microcomputers y sólo existe para ordenadores con los ampliamente difundidos procesadores 8080, 8085 y Z-80. Quien posea un ordenador que trabaje en CP/M tiene acceso a una amplia biblioteca de software de aplicaciones. El CP/M tiene la ventaja de que son precisas (en todo caso) muy pocas variaciones en la transcripción de los programas existentes para que puedan funcionar en otros ordenadores.

Han aparecido ya, de hecho, diversas ofertas de software en CP/M, como por ejemplo WORDSTAR o bien DBASE; estos programas, lamentablemente, aún resultan algo lentos en los ordenadores Amstrad.

En CP/M existen tablas de salto bien definidas para determinadas subrutinas (por ejemplo para la salida a pantalla, etc.). Estas son puestas en funcionamiento por los programas en CP/M, de modo que sólo se precisa un ajuste mínimo. Las rutinas básicas correspondientes son cargadas en cada ordenador mediante la activación de CP/M.

Otra ventaja adicional es la de que el usuario, que ha aprendido la utilización del CP/M puede trabajar con cualquier otro ordenador con este sistema operativo. Quien aprendió BASIC con el ordenador X, no puede programar en BASIC en el ordenador Y con pleno rendimiento. Este no es el caso del CP/M, que está fuertemente estandarizado. En su disco se halla el CP/M 2.2. Lo precisa, en cualquier caso, para formatear o copiar diskettes, pero también puede usarse para otras aplicaciones; si desea aprender CP/M para ello, en las librerías encontrará gran variedad de literatura.

1.3.2 Carga y puesta en marcha del CP/M

Una advertencia previa: Cuando trabaje con su floppy, conecte primero la unidad de disco, y luego el monitor y el ordenador. En caso contrario, en una comprobación interna en la que se reconocen todos los aparatos desconectados, el floppy se registraria como tal y todas las órdenes en un principio dirigidas a él, irían a parar a la unidad de cassette.

Como usuario del Amstrad CPC 6128 recibió, junto al CP/M 2.2 el CP/M 3.0 (plus). La diferencia entre CP/M 2.2 y CP/M 3.0 es, fundamentalmente, la antigüedad: CP/M 3.0 es la más reciente y reelaborada versión de CP/M 2.0. Con CP/M 2.2, por otra parte, "sólo" resulta posible direccionar 64 KBytes, mientras que con CP/M 3.0 puede usted direccionar 128 KBytes. Además, CP/M 3.0 es algo más cómodo que su hermano mayor. Puede, por supuesto, utilizar también CP/M 2.2 en su CPC 6128 - por lo que nos centraremos en el presente libro en esta versión más "sencilla" del CP/M. Una explicación adicional del CP/M 3.0 rebasaría a buen seguro los límites de este libro. Por lo tanto, cuando en lo sucesivo mencionemos el disco de sistema, nos estaremos refiriendo al disco de sistema del CP/M-2.2.

Inserte pues el disco de sistema en la unidad de disco, de modo que la etiqueta sea visible desde fuera y la flecha con la inscripción "CP/M" apunte hacia arriba. Ahora introduzca ICPM.

(Nota: Con I representamos la barra vertical que usted obtiene pulsando simultáneamente Shift y la tecla @.)
Por fin se carga en el ordenador el CP/M desde el disco.

Obtendrá el siguiente mensaje:

CPM 2.2 - Amstrad Consumer Electronics plc.
A>

A partir de ahora dejan de ser operativos los comandos BASIC. Pruébelo introduciendo por ejemplo:

run

A lo que el CP/M contestará mediante

RUN?

lo que significa tanto como ''pues el comando RUN no lo conozco''. Ya habrá notado que en la primera columna se halla un "A>". Este es el llamado símbolo indicador del sistema, mediante el cual se indica que el ordenador está esperando sus órdenes y que está conectada la unidad de disco A. Si sólo posee una unidad de disco obtendrá siempre este mensaje, si es usted propietario de dos unidades, existe además el símbolo indicador B>.

Pero inténtelo ahora con una orden comprensible para el CP/M:

dir

Obtendrá instantáneamente en pantalla el índice del contenido del diskette. dir proviene de la palabra inglesa para índice del contenido, Directory.

1.3.3 Formateado de diskettes

Formatear significa preparar un diskette crudo para la unidad de disco. Un diskette no formateado es para el floppy como una hoja de papel sobre la que hubiéramos escrito con tinta blanca.

Para formatear un disco ha de entrar, desde CP/M, la siguiente orden:

format

En pantalla aparece:

Please insert disc to be formatted into drive A
then press any key

Retire el disco de CP/M e introduzca el disco que deseé formatear. Al formatear se pierden todas las informaciones eventualmente almacenadas en un disco, es decir, al formatear un disco, que ya había sido formateado y en el cual había usted almacenado programas u otras cosas, todo ello se pierde. Por ello se pide mucha atención al formatear ya que una vez formateado un disco, el comando no debería volver a ser llamado.

Cuando haya cambiado el disco, pulse cualquier tecla. El formateado comienza inmediatamente. Cada cara del disco es formateada con cuarenta pistas ordenadas concéntricamente al agujero del eje motriz, de la pista 0 a la pista 39. La pista 0 es la más exterior, la pista 39 la más interior. Además de las pistas existen otras subdivisiones del disco, los sectores. El disco está dividido simultáneamente en 9 sectores, comparables a los trozos de una tarta.

Sólo se puede leer del disco sector a sector. La división en pistas y sectores se da en la mayoría de los sistemas de diskettes.

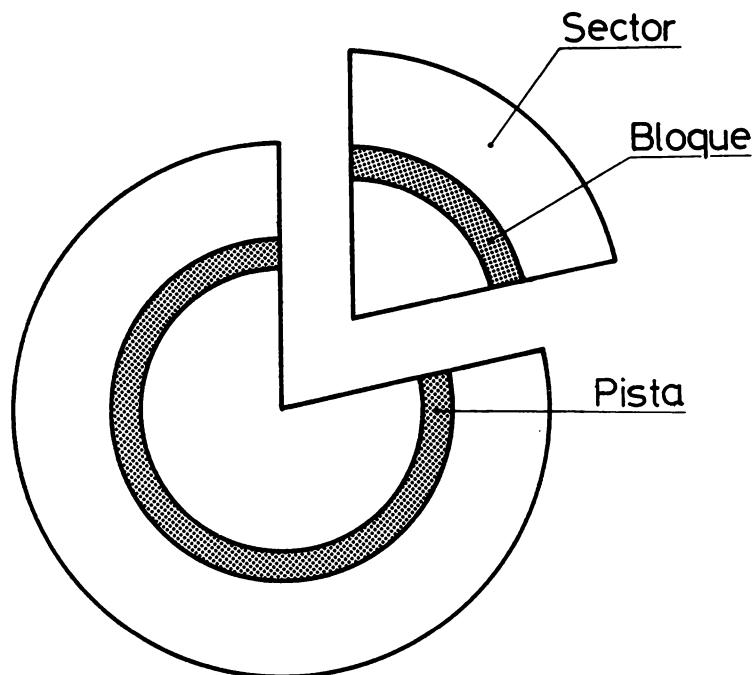


Figura 2. Pistas, bloques y sectores

En general hemos de diferenciar los siguientes conceptos: pista, sector, bloque y record.

Una pista es, como ya se describió, un cuarentaavo del disco. Cada pista tiene 9 sectores, cada uno de los cuales comprende 512 Bytes.

Pero para CP/M existe otra subdivisión, los llamados records. Un record comprende exactamente 128 Bytes. Un sector consta por tanto de 4 records. Esta división resulta necesaria por motivos de compatibilidad con CP/M.

El AMSDOS reconoce todavía una última posibilidad de subdivisión; los bloques. Un bloque comprende 1024 Bytes, y consta por tanto de 2 sectores. Un bloque es la unidad mínima a la que podemos referirnos en BASIC.

Volvamos al tema en si, el formateado.

Tras haber sido formateadas las cuarenta pistas, el sistema vuelve a manifestarse mediante:

Do you want to format another disc (Y/N):

Si desea formatear, p. ej, la otra cara del disco, responda con **y** por "yes=si"; pero también puede formatear cualquier otro disco si lo desea.

El formateado puede repetirse tantas veces como se desee hasta que se responda a la pregunta de repetición con **n** por "no=no"; con ello el sistema le requerirá mediante:

**Please insert a CP/M system disc into drive A
then press any key**

Ahora puede usted pulsar cualquier tecla, es decir, no ha de cambiar previamente el disco ya que el sistema ha situado el sistema CP/M en las dos pistas más exteriores al formatear

(pero no ha incluido los ficheros de comando como por ejemplo la orden format).

Cuando desee formatear un disco, éste ha de estar desprovisto de la protección de escritura. En caso contrario obtendrá el mensaje:

Drive A: disc is write protected
Retry, Ignore or Cancel?

Este mensaje le hace saber que el formateado no puede llevarse a cabo regularmente debido a la protección de escritura. Existen otros mensajes de error similares en los que puede elegir igualmente entre las opciones:

- a) Retry Si desea volver a intentarlo, pulse la tecla "r".
- b) Ignore Si desea ignorar el mensaje de error del floppy, se sigue adelante con el trabajo. Pulse para ello la tecla "i". Pero esto es aconsejable muy raras veces, ya que en los comandos siguientes pueden aparecer unos efectos inesperados.
- c) Cancel Pulsando la tecla "c" se interrumpe el proceso. También en nuestro caso debería usarlo, y retirar la protección cuando se le pida.

1.3.4 Copiar en CP/M

En CP/M es muy sencillo copiar el contenido de todo un disco a otro. Si sólo dispone de una unidad de disco, introduzca el comando

disccopy

Si dispusiera usted de dos unidades de disco, podria emplear la orden copydisc que trabaja a mayor velocidad.

Tras haber introducido discscopy el ordenador se manifiesta mediante:

Please insert source disc into drive A
then press any key

En caso de que obtenga en pantalla el mensaje

FILECOPY?

ello significa que no tiene el disco de CP/M en su unidad de disco.

Extraiga el disco de CP/M de la unidad. Pero si desea copiar el disco de CP/M en si, lo cual deberia hacer al menos una vez, deje el disco de CP/M en la unidad y pulse una tecla.

Copying started
Reading track 0 to 7

El ordenador lee ahora las 8 primeras pistas (=tracks) y las guarda en memoria. Cuando haya acabado con ello obtendrá usted el mensaje:

Please insert destination disc into drive A
then press any key

Extraiga ahora el disco fuente de la unidad e introduzca usted el disco en el que han de ser copiados los ficheros. En caso de que este ultimo disco no esté formateado o bien lo esté incorrectamente, resulta previamente formateado. De igual modo todos los datos que se hallen en el disco

resultan sobreescritos, dado que se elabora una copia completa del disco fuente. De hecho, en cuanto a contenido, la copia no puede distinguirse del original.

Tras introducir el disco y pulsar cualquier tecla obtendrá el mensaje:

Writing track 0 to 7

Repita el mismo proceso para las pistas (tracks) 8 a 15, 16 a 23, 24 a 31 y 32 a 39. El disco habrá sido totalmente copiado, con lo que obtendrá el mensaje

Do you want to copy another disc? (Y/N):

Si no desea copiar ningún disco más, introduzca N y siga las instrucciones de la pantalla. En el caso de desear otra copia, el proceso es idéntico al relatado.

1.3.5 Copiar con dos unidades de disco

Único puede utilizar la orden COPYDISC cuando disponga de dos unidades de disco. El modo de trabajo es similar al del comando DISCCOPY anteriormente descrito. La ventaja de esta orden es, como ya se mencionó, la de que no ha de estar cambiando constantemente los discos fuente y destino.

También en este comando COPYDISC se formatea automáticamente el disco objeto si es necesario. Por lo demás, simplemente ha de ir siguiendo las instrucciones de la pantalla.

1.3.6 DISCCHK - Comprobaciòn de discos

Tras elaborar una copia puede usted comprobar si todo ha sido correctamente copiado; seria irritante, con el disco original estropeado, que tambièn su copia fuese defectuosa, (lo que siempre puede ocurrir). Para comprobar los discos fuente y copia aplique el comando

discchk.

Siga despuès los comandos de pantalla que le reclamaràn la introducciòn de los discos original y copia. Si se comprueban diferencias entre el disco original y el disco copia, obtendrà el mensaje

**Failed to verify destination disc correctly:
track x sector y**

Ello no interrumpe la comparaciòn de ambos discos, de modo que tambièn se indicaràn las diferencias posteriores que pudieran existir. Se comprueban siempre 8 pistas en secuencia. Si posee dos unidades de disco, la comparaciòn es considerablemente mès ràpida y sencilla. Utilice en tal caso el comando

chkdisc

Tan sólo ha de introducir los discos fuente y de copia siguiendo las instrucciones en pantalla, y ambos discos son comparados exactamente igual que con el comando discchk.

De hecho, puede interrumpir todos los comandos CP/M pulsando simultàneamente [Ctrl] y la tecla "C". Si por ejemplo introdujo por error la orden FORMAT, mediante [Ctrl]-C vuelve usted al modo de estado de entrada del CP/M.

He aquí otras funciones de control:

[Ctrl]C	= interrupción de la orden actual
[Ctrl]S	= retiene la salida a pantalla. Pulsando cualquier tecla puede usted reanudar la salida.
[Ctrl]P	= se comunica la salida a impresora, o sea que la salida a pantalla pasa a impresora
[Ctrl]Z	= fin de texto. Se utiliza, por ejemplo, en entradas de texto.

1.3.7 Formatos especiales de formateado

Si desea formatear un disco exclusivo para datos, o bien si precisa un disco exclusivamente para el almacenamiento de programas BASIC, no precisa copiar simultáneamente el sistema operativo CP/M. El CP/M es dispuesto en las pistas 0 y 1. En un disco de datos no necesita usted el CP/M, con lo que se despilfarrarian estas dos pistas. Para evitarlo, podemos servirnos de un comando especial:

format d

Si introduce este comando en lugar de FORMAT, el disco es formateado sin el sistema operativo CP/M. Usted dispone, por tanto, de más espacio para datos y/o programas.

Existen las siguientes posibilidades de formateado:

```
format      formateado con copia de CP/M
format d    formateado en formato de datos sin CP/M
format i    formateado en formato IBM
format v    formateado en formato Vendor
```

Se formatea siempre el disco que se halle en la unidad de disco estàndar. Lamentablemente no resulta posible formatear un disco que se halle en otra unidad.

El formato más usual para usted es el primero si trabaja con CP/M o bien el segundo si trabaja casi exclusivamente en BASIC bajo AMSDOS. Aquí se formatean 9 sectores por pista. En el formato del sistema se numeran los sectores mediante #41 hasta #49 (hexadecimal). Las pistas reservadas tienen la siguiente disposición:

Pista 0,sector #41	:Boot-sector
Pista 0,sector #42	:Sector de configuración
Pista 0,sector #43-#47	:no utilizado
Pista 0,sector #48-#49	:al igual que:
Pista 1,sector #41-#49	:CCP y BDOS

CCP = Console Command Processor.

BDOS = Basic Disk Operating System.

Ademàs, en los primeros sectores de la pista dos se halla el directorio del disco. Puede ver la disposición de los primeros bloques de un disco en el siguiente gráfico:

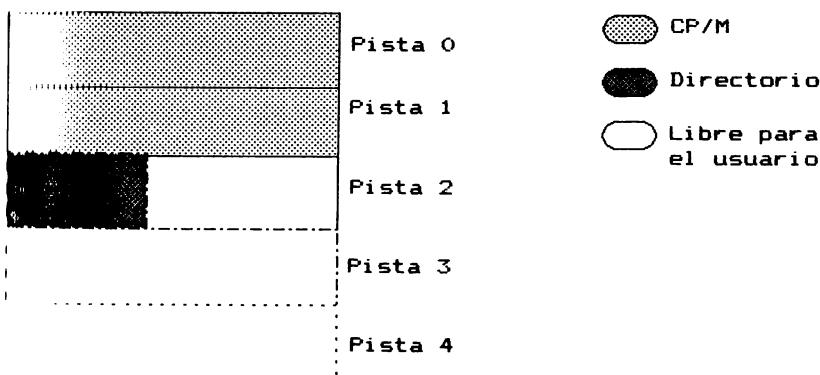


Figura 3. Disposición disco

El formato Vendor es idéntico al formato del sistema excepto en que el sistema operativo CP/M no es simultáneamente copiado. Este formato es utilizado en la venta de software, ya que no se puede vender el CP/M incorporado al mismo.

El formato DATA-ONLY

El formateado (el físico) es el mismo que en el formateado del sistema. Para formatear un disco de datos, use usted el comando FORMAT D. Se formatean 40 pistas de 9 sectores cada una. Los sectores se numeran de #C1 a #C9. El CP/M no es copiado en este caso, de modo que las dos pistas más exteriores, las pistas 0 y 1 quedan disponibles para el usuario. Así dispone usted de $2 \times 9 \times 512 = 9216$ bytes más.

El formato IBM

Se formatea con 8 sectores (#1 - #8) por pista y se reserva una pista. Lógicamente, este formato es el mismo que el formato de discos utilizado en el PC de IBM bajo CP/M.

Este formato deberia utilizarse sólo para usos específicos (por ejemplo, para transferir datos a ordenadores IBM), en cualquier otro caso no es aconsejable.

1.3.8 Órdenes en tránsito y residentes

En CP/M cabe distinguir entre las órdenes en tránsito y las residentes. Las órdenes residentes están disponibles automáticamente al inicializar el CP/M (es decir, al poner en funcionamiento el sistema). En consecuencia, residen formalmente en memoria al igual que antiguamente los reyes en castillos.

Se trata de las siguientes órdenes:

SAVE, A:, B:, DIR, ERA, REN, USER y TYPE.

La mayoría, con mucho, se halla en el grupo de las órdenes en tránsito. Las órdenes en tránsito han de ser cargadas de disco antes de que el ordenador pueda ejecutarlas. Se trata de las siguientes órdenes:

AMSDOS, BOOTGEN, CHKDISK, CLOAD,
COPYDISK, CSAVE, DISCCHK, DISCCOPY,
FILECOPY, FORMAT, SETUP y SYSGEN.

Estas órdenes funcionan exclusivamente en el Amstrad CPC; otros productos utilizan programas auxiliares similares y con nombres parecidos o idénticos. Otros comandos en tránsito estandarizados son:

ED, MOVCPM, PIP y STAT.

Hallará estos ficheros en su disco de sistema CP/M. Los nombres de fichero están marcados mediante un .COM. A título de ejemplo, puede usted hallar en su disco el fichero

FILECOPY.COM (mediante el comando DIR). No dispondrá de las órdenes en tránsito si no fueron copiadas mediante DISCCOPY, COPYDISC, FILECOPY, o PIP.

Así que para tener una copia completa de su disco de CP/M, ha de copiarla mediante

DISCCOPY (para ordenadores con un floppy) o
COPYDISC (para poseedores de dos floppys)

o bien copiando mediante

FORMAT y luego FILECOPY*.*

todos los ficheros en tránsito. Otra posibilidad consiste en copiar sólo los ficheros en tránsito que precise más frecuentemente. Así puede dejar a un lado por ejemplo las órdenes PIP, COPYDISC y CHKDISC, si sólo dispone de una unidad de disco, con lo que tendrá más espacio en su disco.

1.3.9 Copia de ficheros mediante FILECOPY

Para copiar ficheros individuales, sirvase del comando

FILECOPY

En caso de que disponga de dos unidades de disco puede utilizar también la orden CP/M PIP.

Si desea copiar todos los ficheros de un disco, introduzca

FILECOPY *.*

Un nombre de fichero consta de dos partes separadas entre sí por un punto ". ". La primera parte es el nombre de fichero propiamente dicho; la segunda parte es el indicativo de fichero o extensión. La primera parte puede tener una longitud máxima de 8 caracteres, la segunda de hasta 3 caracteres. Esta regla es válida no sólo bajo CP/M en su unidad de discos FDD sino también bajo AMSDOS. Tenemos, por ejemplo los siguientes tipos de indicativos:

Tipo definido sin más precisión. Podría ser por ejemplo un fichero creado en BASIC, abierto sin tipo de fichero mediante OPENOUT.

- .BAS Aquí se trata de un programa BASIC almacenado con el comando BASIC SAVE "Programa", SAVE "Programa",P o bien SAVE "Programa.bas",a.
- .BIN Almacenamiento de una zona de memoria o bien de un programa en lenguaje de máquina.
- .BAK Aquí se trata de un backup, una copia de seguridad. Si almacena por ejemplo un programa bajo AMSDOS y usa un nombre bajo el cual había sido almacenado un programa, el AMSDOS crea automáticamente un fichero backup. Y ello por motivos de seguridad, para no sobreescribir inadvertidamente un fichero. Pero si almacena otra vez el programa bajo el mismo nombre, el antiguo fichero backup es borrado , el último fichero pasa a backup y su nuevo programa es almacenado bajo el nombre de "nombre.bas".
- .COM Se trata de ficheros de órdenes, en los que hay almacenadas órdenes. Todos los programas auxiliares de CP/M serán por tanto de este tipo.
- .DAT Se trata de un fichero creado, por ejemplo, por un programa de gestión de ficheros (como el de este libro) bajo AMSDOS.

.SEQ El fichero es del tipo "secuencial". El tema de los ficheros secuenciales será tratado más adelante con mayor profundidad.

Por lo demás resulta imaginable cualquier otro indicativo de fichero, ya que cualquier combinación posible de 3 letras está permitida, es decir, se puede elegir libremente. En cualquier caso, los tipos de fichero aquí mostrados son los que con más frecuencia se emplean.

Fundamentalmente existen tan sólo dos modos de almacenamiento en los que se basan todos los demás. A saber:

- a) el almacenamiento secuencial y
- b) el almacenamiento relativo de datos.

De modo estandar, el FDD dispone únicamente del más sencillo, el almacenamiento secuencial de datos. En el capítulo 1.6 le aclararemos lo que se entiende por almacenamiento relativo de datos, y su funcionamiento.

1.3.10 Las "wild cards" o comodines

Los asteriscos (*) empleados en el comando FILECOPY*.* son las denominadas wild cards (=cartas salvajes). Estas wild cards pueden hacerle a uno la vida más fácil. ¿Pero qué son wild cards? Bueno, de hecho hay dos tipos de wild cards, el asterisco y el interrogante. Si copia un fichero mediante el comando FILECOPY MATES.BAS, el programa FILECOPY cargará el fichero MATES.BAS en memoria y lo grabará posteriormente en el disco destino con exactamente el mismo nombre. Ha indicado usted por tanto exactamente el fichero que había de ser copiado. Pero frecuentemente aparecen aplicaciones en las que usted quiere copiar muchos ficheros distintos de nombres similares (o bien simplemente referirse a ellos). Supongamos que desea copiar exclusivamente todos los programas BASIC, o sea los programas que finalizan con .BAS. Para ello tendría que observar el directorio, anotarse todos los ficheros que terminen en .BAS y copiar éstos, uno detrás de otro, mediante el comando FILECOPY nombre.BAS. Esto resultaría realmente engoroso. Para aligerar este trabajo existen, afortunadamente, las wild cards, que no son exclusivas del CP/M.

En el caso de nuestro ejemplo tendría que accionar el siguiente comando:

FILECOPY *.BAS

y se copiarían todos los programas BASIC. El asterisco significa aquí: "se cual sea la primera parte del nombre del fichero, si se trata de un programa BASIC, copia".

Así que se copian todos los programas BASIC.

También resulta posible encadenar un asterisco tras uno o más caracteres. A título de ejemplo

FILECOPY f*.*

significa copia todos los ficheros que comiencen por "f", independientemente del tipo de fichero. Con ello los ficheros

"fourier.bas"
"felipe.seq"

serían copiados sin reparo alguno. Existe otra wild card, el interrogante. Un interrogante en cualquier posición sustituye una letra.

Así que en lugar de FILECOPY f*.* podíamos haber escrito FILECOPY f???????.??? o FILECOPY f*.??? o bien FILECOPY ??????.*. Si almacenó, por ejemplo, una agenda en disco, con la orden

FILECOPY 05-??.*

puede copiar todas las fechas del mes de mayo de anualidades anteriores, suponiendo, claro está, un formato de ese fichero adecuado a ello.

FILECOPY 1?1 copiaría todos los ficheros con un uno en la primera y en la tercera (y a la vez última) posición, en medio puede darse cualquier carácter válido.

En la utilización del comando FILECOPY *.* se le pregunta si desea copiar todos los ficheros o bien si desea seleccionarlos. Le son mostrados todos los ficheros uno a uno, y puede decidir (con y para si y n para no), si tal o cual fichero ha de ser copiado o no.

1.3.11 El comando DIR

Mediante el comando DIR puede usted dar salida al índice del contenido del disco. Puede, además, llevar a cabo una selección, es decir, puede excluir determinados ficheros. Esto se consigue mediante la utilización de wild cards.

Al omitir paràmetros se toma *.*. El comando DIR tiene el siguiente comportamiento:

DIR	: muestra todos los ficheros
DIR B:	: muestra todos los ficheros del floppy B:
DIR *.BAS	: muestra sólo los ficheros BASIC
DIR FILECOPY.COM	: muestra sólo el fichero FILECOPY.COM, si existe

Los ficheros se muestran en el mismo orden en el que fueron introducidos, es decir, el primer fichero que fue almacenado en el disco aparece el primero en el índice del contenido del disco.

1.3.12 El comando ERA

Mediante el comando ERA (por ERAse) puede borrar ficheros del disco. En cualquier caso, sólo se borran las anotaciones en el directorio del disco, los datos de los ficheros se conservan - pero de hecho no se puede volver a acceder a ellos. Puede utilizar las wild cards para borrar varios ficheros. Si introduce *.*, la orden ha de ser confirmada ya que ello borraría todas las anotaciones del disco. En caso de que se halle un fichero que sólo sea legible (ver también 1.3.16), la orden se interrumpe.

Comandos posibles:

ERA DISCCOPY.COM	: borra el fichero DISCCOPY.COM
ERA *.SEQ	: borra todos los ficheros marcados .SEQ

1.3.13 El comando REN

Mediante el comando REN (por RENAME) se modifican los nombres de ficheros.

REN nuevo nombre=viejo nombre

En caso de que el nuevo nombre ya exista o bien en caso de que el viejo nombre no exista, aparece un mensaje de error.

1.5.14 El comando TYPE

Con la orden TYPE puede dar salida a pantalla al contenido de ficheros. Si no se trata de ficheros ASCII, como por ejemplo programas BASIC, pueden aparecer símbolos gráficos y similares en pantalla. De aquí que resulte posible variar, por ejemplo, el color de fondo o el modo del cursor.

TYPE EX1.BAS Representación del programa ejemplo EX1

Fig. 15 Cambio de unidad estàndar de disco

Mediante los comandos A: y B: puede usted, mientras disponga de dos unidades de disco, comutar entre ambas.

III La unidad B es la unidad estàndar

El símbolo indicador del sistema también se cambia
consecuentemente.

1.3.16 El comando PIP

Mediante el comando PIP puede usted establecer la transferencia entre ordenador y periferia, o bien copiar en caso de que posea dos unidades de disco.

La sintaxis es la siguiente:

PIP <objeto>=<fuente>.

La <fuente> y el <objeto> pueden ser ficheros o elementos de periferia. Puede referirse a las siguientes unidades de periferia:

Como fuente:

CON: consola = teclado
RDR: interfase serie

Como destino:

CON: consola = pantalla
PUN: interfase serie
LST: impresora

Ejemplo:

Si desea crear un fichero, a introducir por teclado:

PIP Texto.Txt=CON:

Todo lo que introduzca será almacenado en el fichero Texto.Txt hasta que pulse las teclas '[Ctrl]-Z'. Tras ello se cierra el fichero.

El comando 'PIP' no puede ser utilizado para la copia de ficheros con una única unidad de discos. Sirvase para ello del comando FILECOPY.

Otros ejemplos:

PIP LST:=EX2.BAS	Saca el fichero BASIC EX2.BAS por impresora
PIP CON:=EX2.BAS	Lleva el fichero BASIC EX2.BAS a pantalla. La orden es similar al comando TYPE EX2.BAS

1.3.17 El comando STAT

STAT es por status, estado. Por una parte, mediante el comando STAT se puede conseguir una cómoda salida de informaciones de ficheros, similarmente al comando DIR.

STAT
STAT B:
STAT *.BAS

serian ejemplos para ello. Por otro lado, el comando le ofrece una opción muy importante y práctica. Puede conseguir que un fichero sea de "sólo lectura", es decir, que resulte imposible borrarlo o sobreescibirlo inadvertidamente. Mediante el comando

STAT *.BAS \$R/O

conseguirà que todos los ficheros BASIC estén provistos del estado de sólo lectura. A partir de ese momento resulta imposible borrarlos o sobreescibirlos inadvertidamente.

Para la desactivaciòn de este estado incorpore los caracteres para escritura/lectura (\$R/W) en la orden de salida. Para desactivar el anterior comando introduzca:

STAT *.BAS \$R/W

Por otro lado, puede proveer a un fichero del estado de sistema. Tras ello, tal fichero no podrà ser listado mediante el comando DIR, pues es pràcticamente invisible. Mès aùn, dicho fichero ya no podrà ser copiado. Unicamente podr listarse con el comando STAT. Si introduce, por ejemplo:

STAT *.COM \$SYS

Todos los ficheros COM serán provistos del estado de sistema, y ya no podrà copiarlos ni listarlos mediante el comando DIR. En cualquier caso, estos comandos siempre podràn volver a ser llamados.

La inversiòn de esta orden tiene el siguiente aspecto:

STAT *.COM \$DIR

Este comando fija el "estado de directorio".

He aquí algunos comandos STAT muy eficaces:

STAT dsk: Le proporciona todas las informaciones importantes sobre el formato del disco

STAT val: Le proporciona una lista de las abreviaturas y còmo estàn ocupadas en este momento. P.ej. CON:=TTY etc.

o bien 'STAT dev:' y 'STAT usr:'

1.3.18 BOOTGEN

Mediante BOOTGEN resulta posible copiar las pistas 0 y 1 en otro diskette. Puede utilizar este comando cuando desee proveer con CP/M a un disco formateado en formato Vendor o bien cuando desee copiar un nuevo sector de configuración en varios discos.

1.3.19 MOVCPM y SYSGEN

Mediante el comando MOVCPM puede desplazar CP/M a otra zona de memoria. Ello es necesario con frecuencia porque el CP/M colisiona con cierto Software. Puede desplazar el CP/M en pasos de 256 bytes. La sintaxis es la siguiente

MOVCPM <magnitud>*

El valor de la magnitud está comprendido entre 64 y 179. El CP/M estàndar fue generado con la magnitud 179. A título de ejemplo, con MOVCPM 178* desplazará el CP/M hacia abajo en 256 bytes.

Tras ello puede almacenar el CP/M recién generado, bien mediante el comando SYSGEN, o bien en un fichero. Ello es posible también usando MOVCPM.

Inscriba luego el resultado de un comando MOVCPM, mediante SYSGEN, en la pista del sistema. En ello tenemos también tres comandos diferenciables:

SYSGEN*

Este comando escribe en las pistas de sistema el CP/M generado mediante un comando directo MOVCPM previo.

SYSGEN <nombre de fichero>

Con este comando se lee el fichero generado mediante MOVCPM bajo el nombre <nombre de fichero> y se copia en las pistas del sistema. Ejemplo: SYSGEN CPMEXTRA.COM

SYSGEN

En ausencia de paràmetros, se comparan los disco fuente y objeto y se copia correspondientemente el CP/M en el disco objeto. Con esta orden puede usted copiar CP/M en un diskette Vendor.

1.3.20 SETUP

Mediante este comando puede modificar a su antojo el aspecto y el desarrollo del CP/M. Este comando le permite el acceso a la disposicòn del teclado, al acceso a discos y mucho mès. De hecho, sólo deberia utilizar éste comando tras informarse exhaustivamente acerca de los efectos de su uso. En los capítulos 3.7.3.2 y 1.5 del manual del usuario del floppy FDD se halla el proceso correspondiente.

1.3.21 AMSDOS

Mediante el comando AMSDOS desconecta el CP/M y pasa al AMSDOS. Puede usted programar ahora en BASIC, como siempre. Tiene a su disposicòn los comandos del AMSDOS.

1.4 Trabajo en AMSDOS

1.4.1 ¿Qué es AMSDOS?

AMSDOS significa "AMStrad Disc Operating System". AMSDOS apoya el trabajo con el diskette bajo BASIC. El AMSDOS sólo es activo cuando la unidad de disco está conectada al ordenador y encendida - lo que, obviamente, siempre se da en el CPC 664 y CPC 6128.

¡Observe al pie de la letra la secuencia de encendido! Ha de encenderse en primer lugar el floppy; sólo después de ello pueden ser encendidos el monitor y el ordenador. Ello es debido a que, al ser encendido, el ordenador realiza automáticamente una comprobación de los periféricos (impresora, floppy, etc.) que están conectados. En caso de que el floppy todavía esté apagado al encender el ordenador, órdenes posteriores, que deberían ir al floppy, son sin embargo enviadas a la unidad de cassette. Ello es por supuesto válido sólo para el usuario del CPC 464; el CPC 664 y el CPC 6128 reconocen inmediatamente la unidad de disco incorporada.

Todas las órdenes que siguen, que en el CPC 464 van normalmente al cassette, son enviadas al floppy si no se dan otras órdenes respectivas:

```
load "nombre fichero"
run "nombre fichero"
chain "nombre fichero"
merge "nombre fichero"
chain merge "nombre fichero"
save "nombre fichero"
openin "nombre fichero"
closein
openout "nombre fichero"
closeout
```

```
cat
eof
input #9
line input #9
print #9
write #9
list #9
```

Sin embargo, la orden SPEED WRITE se refiere como antes a la unidad de cassette, dado que en la unidad de discos ni la velocidad de escritura ni la de lectura son variables (ver capítulo 1).

Además de las órdenes arriba relacionadas, existen las siguientes órdenes llamadas externas. Se trata de órdenes externas ya que estas órdenes están almacenadas en la ROM del AMSDOS, es decir, en el floppy. Estas órdenes no pueden ser utilizadas en absoluto en el BASIC de cassette, ya que están disponibles a partir del encendido de la unidad de disco. Las órdenes comienzan por I (Shift & Ó); este carácter introduce todos los llamados comandos RSX, o sea, órdenes de ampliación del BASIC. De hecho, los comandos del floppy no son propiamente comandos RSX sino que constituyen una excepción.

```
Ia
Ib
Idir
Idisc
Idisc.in
Idisc.out
Idrive
Iera
Iren
Itape
```

Itape.in
Itape.out
Iuser

Seguro que reconoce los paralelos a diversos comandos CP/M. Ello no ha de extrañarle, ya que en la ejecución de estos comandos RSX se ejecutan exactamente las mismas funciones que bajo CP/M 2.2 ó CP/M 3.0. Los comandos RSX serán aclarados de nuevo mediante ejemplos concretos.

1.4.2 LOAD y RUN

Una de las primordiales funciones de una unidad de discos es el almacenamiento y carga de programas. Si desea cargar un programa de diskette, bien para iniciar lo, bien para programar, ha de utilizar el comando

LOAD "nombre de fichero"

Mediante este comando cargarà programas BASIC del diskette en la unidad de disco.

El ordenador Amstrad reconoce distintos tipos de fichero (ya nos hemos referido brevemente a los diferentes tipos de ficheros). Se precisan distintos tipos de ficheros para distinguir programas BASIC de programas en lenguaje máquina, y para poder diferenciar programas en general de ficheros (por ejemplo, un fichero de direcciones). Ello redundará en última instancia en una mayor claridad para usted, la unidad de disco puede realizar esta distinción también de modo interno, sin ayuda del tipo de fichero indicado en el nombre de fichero.

No sabe usted que el tipo de fichero BAS ha de ser un programa BASIC. El tipo BIN señala un programa en lenguaje de máquina y el tipo de fichero SEQ delata ficheros adicionales.

Si ha introducido el comando LOAD, el AMSDOS intenta cargar el fichero "nombre de fichero....". Si dicho fichero no existe, se intenta cargar el fichero "nombre de fichero.BAS". Si tampoco existe este último, se realiza otro intento, se busca el fichero "nombre de fichero.BIN". Sólo en caso de que tampoco se halle este fichero, aparece un mensaje "File not Found". Pero también puede introducir

LOAD "nombre de fichero.BAS"

para excluir el que el AMSDOS cargue un fichero eventualmente existente bajo el nombre de "nombre de fichero. ".

En caso de que sea el feliz propietario de una segunda unidad de discos, sería fabuloso poder leer programas también de esta unidad de disco en la memoria del ordenador. Para conseguirlo, es preciso incluir la unidad en el nombre de fichero. La primera unidad de disco (la incorporada) tiene el indicativo de unidad de disco 'A:', la segunda unidad de disco (la externa) tiene el indicativo de unidad de disco 'B:'. Si se desea cargar un programa de la segunda unidad, el comando de carga tiene el siguiente aspecto:

LOAD "B:nombre de fichero"

Si el disco no está, o está mal colocado, aparece en pantalla el siguiente mensaje de error:

**Drive A: disc missing
Retry, Ignore or Cancel?**

Introduzca entonces el disco correctamente en la unidad y pulse la tecla "R" por "Retry", es decir repetir.

Observe que no es preciso concretar el tipo de fichero. El AMSTRAD encadena automáticamente la terminación ".BAS" a los nombres de ficheros al cargar y almacenar programas BASIC. En caso de que haya cargado un programa bajo otro nombre con un indicativo distinto al de ".BAS", puede cargar este fichero mediante el comando:

LOAD "nombre de fichero.xxx"

En xxx puede introducir los tipos de fichero correspondientes.

En caso de que no exista el fichero "nombre de fichero" (verbigracia por una introducción errónea del nombre de fichero), obtendrá el mensaje de error:

filename.xxx not found.

Compruebe entonces su nombre de fichero e intételo de nuevo si es necesario.

Si obtiene el mensaje de error Type mismatch, ello le indica que ha olvidado introducir las comillas al comienzo del nombre de fichero.

El mensaje de error

```
Drive A: read fail  
Retry, Ignore or Cancel
```

Indica la aparición de un error de lectura en el disco. Es posible que el disco sea defectuoso (o que haya introducido un disco erróneo). También cabe la posibilidad de que su disco no fuera correctamente formateado en el formato AMSTRAD.

El mensaje

Press PLAY then any key:

significa que la conexión del ordenador a la interfase, es decir al floppy no es correcta. También es posible que no haya encendido la unidad de disco en primer lugar. ¡Si este mensaje apareciera en un CPC 664 o en un CPC 6128, sería realmente para preocuparse! Otra posibilidad es la de que haya utilizado la orden Itape(.in); en tal caso introduzca:

Idisc

e intèntelo de nuevo. Si vuelve a obtener el mismo mensaje, desconecte su ordenador y vuelva a encenderlo.

Al igual que en el BASIC de cassette, existe la posibilidad de ejecutar programas automáticamente tras su carga. También aquí se utiliza para ello el comando:

RUN "nombre de fichero"

Aquí ya no disponemos de la posibilidad existente en el BASIC de cassette de utilizar el comando RUN ", dado que se precisa exactamente el nombre del fichero a cargar. Para la orden RUN "nombre de fichero" son válidos los mismos mensajes de error, y sus consecuencias, que para LOAD.

Precisamente al cargar programas BASIC notará como usuario del CPC 4646 la rapidez de la unidad de disco Amstrad frente a la unidad de cassette.

Por supuesto que puede hacer uso también de las restantes órdenes BASIC de carga. El cómo puede utilizar los comandos

```
CHAIN "nombre de fichero"  
MERGE "nombre de fichero" y  
CHAIN MERGE "nombre de fichero"
```

es mejor que lo mire en el manual, ya que en el proceso de carga pueden suceder distintas cosas con los programas que se hallan en memoria.

Una posibilidad muy útil es la de poder definir, bajo AMSDOS al igual que en CP/M, las llamadas zonas de users. USER proviene del inglés y está por la palabra española "usuario". Si ya se ha mirado un directorio, habrá observado que la primera línea tiene el siguiente aspecto:

```
Drive A: user 0
```

Puede definir números user de 0 a 15, siendo el estàndar el número user 0. Mediante el número user puede referirse a diferentes directorios, es decir, puede dejar por ejemplo como user 0 las órdenes CP/M y bajo user 1 sus programas BASIC y datos. Si desea cargar un programa del user 1, dispone de dos caminos para ello. El primero consiste en definir el user mediante la orden IUSER. Con

```
IUSER,1
```

lo conseguimos. Despuès puede usted cargar el programa mediante LOAD "nombre de fichero". Tras cargar el programa se hallará usted todavía en el user 1, hasta que lo redefina. La segunda posibilidad consiste en incluir el user en el nombre de fichero. Mediante

```
LOAD "1:nombre de fichero"
```

cargarà usted un programa del user 1, independientemente de su estado de user en ese momento. Ha de poner, por tanto, el número de user y después el doble punto ante el nombre de fichero en si, como en el floppy. Si desea elegir el floppy y el número de user, ha de dar en primer lugar este número, luego el floppy y, finalmente, el doble punto. Ejemplo:

```
LOAD "8B:Nim"
```

carga el juego "Nim" de la unidad de disco B, el user es 8. La sintaxis es, por tanto, la siguiente:

```
LOAD "<num.user><drive>:nombre de fichero"
```

<num.user> significa un número user cualquiera entre 0 y 15, <drive> indica la unidad de disco, es decir, A o B.

1.4.3 Las órdenes de fichero

Para el tratamiento de ficheros con la unidad de cassette o la unidad de disco, se precisa de las órdenes:

```
OPENOUT "nombre de fichero"  
OPENIN "nombre de fichero"  
CLOSEOUT  
CLOSEIN  
INPUT #9  
LINE INPUT #9  
PRINT #9  
WRITE #9
```

El siguiente programa le escribe los primeros 100 números en un fichero en cinta (CPC 464) o en disco (CPC 664 & 6128):

```
10 REM =====
20 REM     Programa ejemplo para ficheros de datos
30 REM =====
40 :
50 OPENOUT "numeros"
60 FOR I=1 TO 100
70 PRINT #9,I
80 NEXT I
90 CLOSEOUT
100 :
110 REM =====
120 REM             Ahora, lectura
130 REM =====
140 :
150 MODE 1 : PEN 1
160 PRINT "Rebobine la cinta"
170 PRINT "y pulse una tecla"
180 D$ = INKEY$ : REM *** borra buffer ***
190 IF INKEY$ = "" THEN 190
200 :
210 OPENIN "numeros"
220 WHILE NOT EOF
230   INPUT #9,I
240   PRINT I,
250 WEND
260 CLOSEIN
270 END
```

¡Atención! Este programa escribirá sobre cassette sólo si posee un CPC 464 y tiene desconectada del ordenador la unidad de disco. No obstante, si posee un CPC 464 y su unidad de disco está conectada al ordenador, incluya el siguiente comando antes de la ejecución del programa:

ITAPE

Todas las órdenes de fichero se dirigen de nuevo a la unidad de cassette. Puede cronometrar lo que tarda el programa en ejecutarse. Cuando haya acabado, introduzca

Idisc

y ejecute el programa de nuevo. Ahora se abre el fichero "numeros" en la unidad de discos. Dado que no puede rebobinar un disco, puede saltarse el mensaje "rebobine la cinta" y pulsar directamente una tecla.

Introduzca un disco que no esté protegido frente a escritura; lo mejor es que utilice uno que haya formateado como ejercicio. Luego introduzca RUN.

Observará que la duración de este proceso es considerablemente más reducida. Así que, en principio, no le costará mucho esfuerzo si desea trabajar con ficheros secuenciales en disco si ya lo hizo en BASIC con la unidad de cassette.

También puede comutar al viejo conocido, el BASIC de cassette, mediante el comando ITAPE, lo que se verá en el capítulo 1.4.4

La función EOF, tal y como aparece en la línea 220 de nuestro programa ejemplo significa End Of File, es decir, fin del fichero. Esta función es falsa(=0) si todavía pueden ser leídos caracteres del fichero abierto para lectura. EOF será cierta=(-1) después de haberse leído el último carácter de dicho fichero; ya no pueden recogerse más caracteres de este fichero. Esta orden es muy importante cuando se leen ficheros secuenciales y el número de caracteres a leer no es conocido previamente. EOF le posibilita por tanto trabajar de un modo muy flexible con ficheros secuenciales.

Ahora nos ocuparemos exclusivamente de órdenes de disco. Las prestaciones de la utilización de cassette son considerables, pero no han de ser exhaustivamente comentadas en un libro sobre floppys.

Introduzca ahora la orden

CAT

Obtendrá en pantalla el índice del contenido del disco o directorio. Por directorio entendemos la lista de los ficheros almacenados en una cara de diskette. El disco ha de ser organizado; existen para ello bloques especiales en el mismo en los cuales están almacenadas las siguientes informaciones: los nombres de los ficheros, sus longitudes, a qué tipos pertenecen y dónde puede hallarlos el DOS, el Disk Operating System.

Si mira el fichero mediante las órdenes CAT o IDIR, obtiene información acerca de qué ficheros se hallan en su disco y de cuánta memoria dispone usted.

Al utilizar el comando CAT se le da suplementariamente la longitud del fichero en KBytes, lo que no sucede con el comando IDIR. La mínima unidad para un fichero es un KByte.

Las órdenes IDIR y CAT no son idénticas. Ambas muestran el directorio, pero el comando IDIR muestra los ficheros en el mismo orden en que fueron puestos en el disco. La orden CAT muestra la longitud de los ficheros, ordenándolos además alfabéticamente antes de darles salida. Como se ve, dos importantes diferencias.

En dicho directorio hallarà tambièn el fichero "numeros" que habíamos creado anteriormente:

numeros . 1 k

Dado que no hemos incluido ningún tipo de fichero, tras el punto tampoco aparece tipo de fichero alguno. El nombre de fichero es automáticamente completado hasta la octava posición mediante espacios.

Observarà que nuestros 100 números ocupan 1KByte, es decir 1024 bytes. Pero incluso en el caso de que hubiéramos escrito en este fichero un único número, el fichero "numeros" ocuparía 1 KByte, dado que esta es la mínima unidad.

Pero ejecute de nuevo su programa y vea lo que ocurre con el directorio. Introduzca RUN, luego CAT.

Verá que hay dos ficheros con el nombre "numeros". Uno de los ficheros carece de indicativo de fichero. En el mismo están los 100 números escritos la vez anterior. Pero existe además un fichero "numeros.BAK". Recordará de la parte de CP/M que BAK es la abreviatura de BAckUp. Al abrir el fichero "numeros" mediante el comando OPENOUT "numeros", el AMSDOS comprobó la no existencia previa del fichero "numeros". Dado que ya habíamos ejecutado nuestro programa, el fichero sí existía. De hecho, ya lo vimos en el directorio. Ya que el AMSDOS no desea sobreescribir este fichero, elabora una copia de seguridad bajo el nombre de "nombre de fichero.BAK". Sólo entonces se borra el antiguo fichero, para que pueda abrirse uno nuevo con el mismo nombre. Aprenderá a apreciar esta peculiaridad del AMSDOS. Cuando pueda prescindir con toda seguridad del fichero backup, puede borrarlo también.

También al almacenar un programa BASIC con el comando SAVE "nombre de fichero" o SAVE "nombre de fichero.BAS", A, un programa eventualmente preexistente es almacenado de modo previo como copia backup, así que la protección frente a sobreescrituras involuntarias es bastante completa. Pruebe ahora lo siguiente, tecleando NEW:

NEW (RETURN)

```
10 PRINT "Primera parte."
20 PRINT
SAVE "Prog"
```

El programa será almacenado en disco bajo el nombre "Prog .BAS". Adjunte las siguientes líneas:

```
30 PRINT "ya completada"
40 PRINT
```

Introduzca de nuevo la orden para almacenar programas:

```
SAVE "Prog"
```

Es decir, empleando exactamente el mismo nombre de fichero de antes. Ahora puede introducir LOAD "Prog.BAK". (Observe que, en la práctica, no ha de constar forzosamente de ocho posiciones al entrar nombres de fichero). Si lista su programa reconocerá que ha cargado la primera versión de nuestro corto programa. Introduzca ahora

```
LOAD "Prog"
```

y será cargada en memoria la última versión con cuatro líneas. Suponiendo que la complete con la línea

```
50 END
```

y vuelva a almacenarla en disco, la primera versión, de sólo dos líneas, habrá desaparecido. Nuestro programa de cuatro líneas lo hallará bajo el nombre de "Prog.BAK", y el actual bajo el de "Prog.BAS".

Si almacenara el programa bajo el nombre "numeros", ello no generaría fichero backup alguno, dado que no existe ningún fichero con el nombre "numeros.BAS". Así que se conservan sus ficheros "numeros.BAS" y "numeros.BAK". Si vuelve a almacenar su programa, el fichero "numeros.BAK" será sobreescrita por una copia del programa BASIC.

Tres posibles los siguientes comandos SAVE:

SAVE "nombre de fichero",A

que almacena el programa como fichero ASCII. Estos ficheros de programa son algo más largos que los ficheros "normales" de programas BASIC, dado que las distintas órdenes, como por ejemplo "PRINT", son escritas en el disco carácter a carácter, y no como tokens. (Un token consta de un carácter y representa una palabra clave BASIC). Los programas así almacenados también pueden ser leídos carácter a carácter a través de un programa.

SAVE "nombre de fichero",P

almacena un programa bajo estado de protección, es decir, dicho programa no podrá ya ser LISTado ni reinicializado o almacenado tras una interrupción mediante (ESC)(ESC).

SAVE "nombre de fichero",B,<dirección inicial>,<longitud>

Con este comando puede almacenarse una zona de memoria, como por ejemplo la memoria de pantalla, como fichero binario. Ha de incorporar la dirección inicial y la longitud de la zona de memoria. Al cargar mediante el comando LOAD "nombre de fichero.BIN" el fichero es cargado en la dirección inicial correspondiente. La figura 4 ilustra este proceso.

Almacenamiento de una zona de memoria

Figura 4. Almacenamiento de una porción de memoria

Naturalmente, en la orden SAVE puede incorporar al fichero, al igual que en la LOAD, indicaciones acerca de la unidad de disco y número de user.

Ya habrá observado que puede trabajar con la orden SAVE exactamente igual a como está acostumbrado del BASIC de cassette. La utilización de las restantes órdenes BASIC de carga y escritura de datos le será explicada en el siguiente capítulo, donde también se aclarará el modo de trabajo con ficheros secuenciales a base de ejemplos.

1.4.4 Las órdenes adicionales del AMSDOS

Al encender la unidad de disco dispone usted de otras órdenes de diskette, almacenadas en la ROM (Read Only Memory) del floppy. A estas órdenes también se les llama órdenes externas, ya que están definidas en el floppy.

Sin la unidad de disco, dichas órdenes no están a su disposición. Puede reconocerlas en que comienzan por el carácter I (Shift & tecla de Ø). Puede utilizar estas órdenes, como comando directo o bien incluidas en programas. He aquí la lista de las órdenes externas con ejemplos para su mejor comprensión.

Cuando un parámetro está situado entre los caracteres < y >, significa que se trata de un sinónimo. Por ejemplo, hallará frecuentemente la expresión <expresión de cadena>, lo que significa que ha de poner en esta posición del comando una variable de cadena. No obstante, si la <expresión de cadena> está entre corchetes [<expresión de cadena>], ello significa que la expresión de cadena puede omitirse, es decir, es opcional.

- IA Esta orden fija la unidad de disco A como unidad estàndar. No la puede utilizar con una única unidad, ya que en tal caso dicha unidad A es, automáticamente, la unidad estàndar. Entendemos por unidad de disco estàndar a la unidad a la que van automáticamente referidas las órdenes de floppy sin especificaciones adicionales.
La orden es idéntica al comando

```
a$="a"  
IDRIVE,0a$
```

- IB Fija la unidad de disco B como unidad estàndar (ver arriba).
Esta orden es idéntica a

```
a$="B"  
IDRIVE,0a$
```

- ICPM Se carga el sistema operativo CP/M del disco. Si utiliza este comando, en la unidad de disco A ha de hallarse un disco que haya sido formateado con CP/M en ambas pistas del sistema. Es el caso de los discos incluidos con el aparato en la compra.

IDIR El comando IDIR, [*<expresión de cadena>*] muestra el contenido del disco y es idéntico a la orden de BASIC estándar CAT.

Le será mostrado el directorio del disco así como la memoria libre del mismo.

En caso de que la *<expresión de cadena>* opcional falte, se toma **.**, es decir, se muestran todos los ficheros (sin selección). Como ya se mencionó, los asteriscos e interrogantes *(*/*?)* son wild cards y representan caracteres en general. Como recordatorio: un interrogante significa 'aquí puede ir cualquier carácter', y un asterisco: 'de aquí al final del nombre de fichero puede ir cualquier combinación de caracteres'. Suponiendo que lo que desee sea simplemente listar todos los ficheros BASIC, puede conseguirlo mediante el comando

```
a$=".BAS"  
IDIR,2a$
```

pero también podría querer listar, por ejemplo, todos los ficheros que comiencen por "mates". Para ello ha de teclear:

```
a$="mates*.*"  
IDIR,2a$
```

La utilización de wild cards (también llamadas comodines) puede resultar muy útil y ahorrar mucho tiempo. En el comando IDIR esto contribuye a clarificar las cosas.

La ventaja de la orden CAT consiste en que ordena alfabéticamente los nombres de los ficheros antes de la salida e incorpora las longitudes de los mismos. La orden IDIR tiene la ventaja de que se puede seleccionar la salida.

IDISC Esta orden engloba las dos órdenes Idisc.in e Idisc.out. Dado que ha de utilizar las mismas órdenes BASIC para las unidades de disco y de cassette, solamente obtendrá respuesta en uno de los aparatos. Para utilizar también el cassette cuando esté trabajando con el Floppy, Con las órdenes IDISC e ITAPE tiene usted la posibilidad de elegir el aparato de entrada o de salida

IDISC.IN Tras dar esta orden, las siguientes órdenes de entrada se refieren, de modo automático, al floppy y no al cassette:

```
LOAD "nombre de fichero"  
RUN "nombre de fichero"  
CHAIN "nombre de fichero"  
CHAIN MERGE "nombre de fichero"  
MERGE "nombre de fichero"  
OPENIN "nombre de fichero"  
CLOSEIN  
EOF  
CAT  
INPUT #9  
LINE INPUT #9
```

Este comando desactiva de nuevo las órdenes ITAPE o ITAPE.IN. Así, por ejemplo, la secuencia

```
ITAPE
IDISC.IN
OPENIN "fichero"
OPENOUT "backup"
```

abriría un canal de entrada en el floppy para lectura, pero el canal de salida escribiría en cassette. Así puede copiar ficheros de disco en cassette (como copia de seguridad).

IDISC.OUT Esta orden es similar a la orden IDISC.IN con la salvedad de que las salidas se dirigen aquí al floppy. Resultan afectadas las siguientes órdenes:

```
SAVE "nombre de fichero"
OPENOUT "nombre de fichero"
CLOSEOUT
WRITE #9
PRINT #9
```

ITAPE

```
IDISC.OUT
IOPENIN "backup"
IOPENOUT "fichero"
```

Esto sería la inversión del ejemplo bajo IDISC.IN. Con las órdenes de lectura se leería ahora del cassette el fichero backup, pero las órdenes de escritura se referirían al floppy.

IDRIVE Esta orden IDRIVE,<expresión de cadena> es idéntica a los comandos IA e IB, pero es flexible por cuanto puede indicar, en una variable de cadena, el nombre de la unidad de disco estàndar elegida.

```
a$="b"  
IDRIVE,0a$
```

tendria como consecuencia que, a partir del momento de la ejecución de la orden, la unidad B es la unidad estàndar por definición. No puede usar este comando si sólo dispone de una unidad de disco.

IERA ERA viene de la palabra inglesa erase=borrar. Mediante este comando puede usted borrar ficheros. En él puede utilizar tambièn wild cards, así por ejemplo la orden

```
a$=".seq"  
IERA,0a$
```

borraria todos los ficheros del tipo "seq". La utilizaciòn de esta orden deberà ser suficientemente meditada, sobre todo si trabajamos con wild cards, dado que podrian ser borrados inadvertidamente ficheros todavia útiles. Se desaconseja la utilizaciòn de comodines en la orden IERA si no se tiene un poco de soltura con el floppy y el manejo de comodines.

El comando

```
a$="*.*"  
IERA,0a$
```

borraria todos los ficheros. En este caso el AMSDOS espera una confirmacin especial del comando.

IREN RENName(= modiﬁcacin del nombre) de un ﬁchero. Con este comando puede, por tanto, cambiar el nombre de un ﬁchero. La primera expresin de cadena ha de contener el nuevo nombre de ﬁchero, la segunda expresin de cadena contiene el antiguo. Si por ejemplo desea cambiar el nombre del ﬁchero "Luisa" porque ha cortado con ella y dejarlo en "Juana", puede introducir las siguientes órdenes:

```
ant$="Luisa.bas"  
nue$="Juana.bas"  
IREN,0nue$,0ant$
```

Si mira el directorio mediante el comando IDIR, observar que el ﬁchero "Luisa.BAS" ya no existe, y que tiene en su lugar el ﬁchero "Juana.BAS". Es obligatorio indicar ambas expresiones de cadena.

ITAPE Este comando desactiva los comandos IDISC, IDISC.IN y IDISC.OUT y establece tanto la entrada como la salida por unidad de cinta. El comando comprende las órdenes ITAPE.IN y ITAPE.OUT.

ITAPE.IN Esta orden utiliza la unidad de cassette como fichero de entrada. La orden desactiva los comandos IDISC y IDISC.IN.

ITAPE.OUT Se utiliza la unidad de cinta como aparato de salida. Desactiva las órdenes IDISC y IDISC.OUT.

IUSER Con este comando tiene usted la posibilidad de definir un user = usuario determinado. Habrá observado que junto al directorio del disco aparece el mensaje

USER:0

Este es un comando CP/M especial y muy útil. Hallará más información al respecto en el capítulo 1.4.2.

Mediante el comando IUSER puede usted ocultar ficheros a otras personas.

Almacene cualquier programa BASIC del siguiente modo:

IUSER,3
SAVE "programa"

```
IUSER,O  
CAT
```

o sencillamente con el comando

```
SAVE "3:programa"
```

No hallará en la lista el programa recién almacenado, ya que dicho programa aún no se halla en la zona actual de usuario. Pero si introduce

```
IUSER,3  
CAT  
IUSER,O
```

Le aparecerá un índice mucho más corto, de un único elemento. Puede crear así distintos directorios o proteger ciertos programas de otros usuarios.

Habrá observado que las órdenes con una expresión de cadena son de utilización bastante incómoda. Así, por ejemplo, ha de introducir

```
a$="a"  
IDRIVE,0a$
```

en lugar del sencillo

```
IUSER,"a"
```

Este sencillo método comentado resulta posible en los ordenadores CPC 464 y CPC 6128, ya que en ellos se llevó a cabo la corrección de los sistemas operativos de BASIC. Si desea escribir programas que puedan funcionar en los tres ordenadores se hace necesario emplear el método aquí comentado, ya que una IDRIVE,"A" por ejemplo, llevaría en el CPC 464 a un mensaje de error - vale por tanto la pena, utilizar este procedimiento un tanto engoroso, con vistas a la compatibilidad.

Y sin embargo, el sistema operativo de BASIC del CPC 464 está provisto de tal modo que se transfiere una dirección de una expresión de cadena al sistema operativo AMSDOS. Este recoge posteriormente de la memoria la cadena por sí mismo. Las cadenas se almacenan en algún lugar de la memoria. Mediante la función Ønombre de variable obtendrá la dirección donde está almacenada la variable, en nuestro caso la cadena. Realmente, la forma de transferencia de datos no es muy cómoda, pero se acostumbrará usted rápidamente.

El comando Ø es un comando BASIC muy útil, que no se menciona en el manual. Introduzca:

```
a=12.2
PRINT Øa
```

Dependiendo del número de variables que haya utilizado previamente, el valor que salga será mayor o menor. Cuanto más tarde sea declarada una variable, mayor es el valor de la función Ønombre de variable. También es significativa la longitud actual del programa BASIC ya que la tabla de variables se halla en la memoria ineludiblemente tras el programa BASIC.

Como debe saber, las variables numéricas se van disponiendo en la memoria de abajo a arriba, las cadenas están en el límite superior de la memoria y se mueven hacia abajo. Puede visualizarlo mediante el siguiente ejemplo:

```
NEW
a=10.1 : b=20 : a$="Ejemplo 1" :
b$="Ejemplo 2"
PRINT @a,@b,@a$,@b$
```

Si observa por fin las direcciones de las variables, verá que las variables numéricas ocupan 9 bytes. Aquí están almacenados los valores de las variables a y b, así como sus nombres. Pero las variables de cadena son almacenadas de otro modo. Introduzca

```
PRINT PEEK(@a$)
```

Obtendrá el valor de 9, lo que es exactamente la longitud de la variable a\$. Lo mismo vale para la variable de cadena b\$. Los valores de las posiciones de memoria

```
@a$+1 así como
@a$+2
```

contienen un puntero o indicador. Este puntero le comunica al sistema operativo dónde puede hallar la cadena. Este método de almacenamiento de cadenas le puede parecer complicado, pero tiene una ventaja decisiva: no ha de desplazarse toda la tabla de variables si una cadena modifica su longitud; simplemente hay que modificar los tres bytes descritos.

Pero démosle salida a nuestra cadena de un modo poco convencional:

```
ad=PEEK(@a$+1) + 256*PEEK(@a$+2)
```

Hemos calculado la dirección en la que está almacenada la cadena de caracteres a\$ - en la variable ad como memoria intermedia.

```
FOR I=0 TO PEEK(0a$)-1
    PRINT CHR$(PEEK(ad+I));
NEXT I
```

Y obtendrá de nuevo el contenido exacto de la variable a\$. Es muy fácil alterar el puntero de la variable de cadena con lo que consigue poder desviar a voluntad el contenido de la variable. Esta propiedad se utiliza en los programas del capítulo 5 (rutina de búsqueda de errores y gestión de ficheros relativos). Como ejemplo, desvie la cadena a\$ a su pantalla:

```
POKE 0a$+1,&00POKE 0a$+2,&C1
```

El puntero para la cadena caracteres apunta ahora a el centro de la memoria de pantalla. Dele ahora salida a la variable de cadena a\$:

```
PRINT a$
```

El efecto no se puede describir con precisión, ya que a cada uno le aparece algo distinto en pantalla, posiblemente parpadee el margen, o varíe el modo de escritura. En cualquier caso obtendrá una oleada de caracteres diversos. Esto debería servirle como introducción a la orden 0.

1.5 Almacenamiento secuencial de datos

1.5.1 ¿Qué es el almacenamiento secuencial de datos?

Por descontado que una unidad de disco no ha de servir exclusivamente para el almacenamiento y carga de programas; resulta especialmente indicada para el almacenamiento de grandes cantidades de datos, que apenas podriamos manejar (o en absoluto) con la unidad de cassette.

Aquí cabe volver a aclarar el concepto de datos. Hemos de distinguir entre ficheros que contienen programas, y ficheros que simplemente contienen datos. Ambos tipos de ficheros son almacenados de modo secuencial (ver abajo), pero se distinguen por una diferencia sustancial:

Usted puede leer los ficheros de programa (bien de programas BASIC, bien de programas en lenguaje de máquina) de modo directo mediante los comandos LOAD directamente en memoria. No es así en los ficheros de datos; éstos sólo pueden ser leídos en memoria y gestionados a través de programas. Así, un programa de contabilidad escrito en BASIC sería un fichero de programa en el disco, mientras que los datos generados por este programa estarían almacenados en el disco en un fichero de datos.

Mediante la DDI-1 usted únicamente puede elaborar de modo estándar ficheros secuenciales, como ya conoce del BASIC de cassette; el principio es idéntico. El almacenamiento secuencial de datos no es el método de almacenamiento de datos más rápido, pero si el más sencillo y completamente suficiente para problemas de magnitud de reducida a mediana. Teniendo en cuenta que además resulta sencillo de entender, puede aprenderse fácilmente.

Secuencial no significa otra cosa que carácter a carácter. Un fichero secuencial es una secuencia de caracteres (letras, cifras, caracteres especiales, etc., es decir una serie de caracteres. Para leer el último carácter de un

fichero han de releerse todos los caracteres anteriores, del primero al penúltimo. Imagíñese un libro de 100 páginas. Desea leer la primera letra de la página 100 y para ello está obligado a releer antes todos los caracteres de las páginas 1 - 99! De este modo se lee un fichero secuencial - un método no demasiado refinado. Por suerte la DDI-1 es muy rápida, de modo que apenas nos damos cuenta de este tipo de cosas.

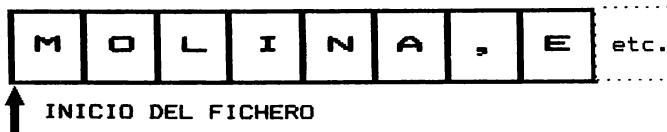


Figura 6. Fichero secuencial

Otra posibilidad, realmente más confortable, de leer datos es la posibilidad de acceder directamente a un fichero, lo cual se denomina acceso directo o bien random access (acceso aleatorio); también se ha impuesto el concepto de ficheros relativos.

Acceso directo significa que - siguiendo con el ejemplo del libro - puede leer directamente, por ejemplo, la letra 55 de la página 29 sin necesidad de leer por encima todas las demás páginas anteriores. Cuando, en un fichero secuencial, alcance el juego de datos número 100, no podrá ya leer un carácter del quinto juego de datos - es decir, no se puede retroceder. Esta es la limitación de las posibilidades de los ficheros secuenciales, limitación que ya no se da en los ficheros relativos.

Pero todo esto es un simple boceto de las diferencias. En el capítulo 5 hallarás más sobre ficheros relativos y su programación en la FDD.

Pasemos a explicar, de la mano de diversos ejemplos, el modo de utilizar con efectividad el almacenamiento secuencial de datos, ya que es (de momento) la única posibilidad que se nos ofrece para almacenar datos en disco.

Vamos ahora a crear nuestro primer pequeño fichero. Como ejemplo confeccionaremos una agenda telefónica, para almacenar los teléfonos más importantes de nuestros parientes o amigos.

Al idear un fichero, se impone en primer lugar la siguiente reflexión: ¿Qué quiero almacenar? En este ejemplo nos hemos decidido por estas anotaciones:

- 1) Apellido (=campo 1)
2) Nombre de pila (=campo 2)
3) Número de teléfono (=campo 3)

Un juego de datos (ficha) es una unidad completa; cada juego de datos contiene ciertas informaciones, que se hallan también en cada uno de los otros juegos de datos. Nuestro juego de datos consta de 3 campos, también llamados campos de datos. Para facilitar la comprensión del concepto de juego de datos, observe el gráfico:

JUEGO DE DATOS

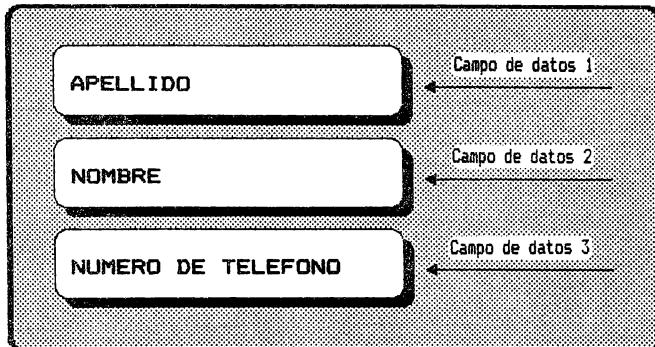


Figura 6. Juego de datos

Podemos disponer así de un juego de datos llamado "Molina", que contiene como información el nombre, apellido y número de teléfono de nuestro colega Molina.

Campo 1: Apellido
Campo 2: Nombre
Campo 3: Número de teléfono

En este gráfico puede reconocer a simple vista las subdivisiones de un juego de datos.

Al escribir los datos en un fichero, no resulta tan fácil reconocer las divisiones de un fichero como aquí, sobre el papel, sino que están separados entre sí simplemente de un modo físico. La separación entre los juegos de datos no se diferencia de la separación entre los campos de datos, se utiliza el mismo carácter separador. La separación de los juegos de datos - es decir, la correcta disposición de los campos de datos - corresponde al programador. El es el único que sabe cuántos campos hay en cada juego de datos. Es por este motivo que dicha información debería incluirse de forma clara en los listados de los programas.

Al finalizar una entrada visualizada por pantalla, usted pulsa la tecla ENTER, también llamada retorno de carro o carriage-return. En el almacenamiento secuencial de datos ocurre exactamente lo mismo, los distintos campos de datos se separan mediante un retorno de carro en el disco, así que mediante la "tecla" ENTER finaliza usted la entrada en un campo de datos.

M	O	L	I	N	A	←	E	M	I	L	I	O	←
---	---	---	---	---	---	---	---	---	---	---	---	---	---

 <Retorno de carro>

Figura 7. Separación de campos de datos

Para facilitarle la visualización de cómo se separan los distintos campos de datos, observe la figura 7. El carácter separador salto de carro se representa mediante la flecha quebrada.

En nuestro ejemplo puede reconocer que los campos de datos no han de tener la misma longitud, sino que pueden tener, en ficheros secuenciales, cualquier longitud, siendo ello además válido para la longitud de los juegos de datos. A pesar de esto, los campos de datos pueden determinarse exactamente, dado que están separados mediante el símbolo de retorno de carro.

Para leer un fichero de este tipo, se utiliza la orden INPUT #9.

```
OPENIN "nombre de fichero"  
INPUT #9,apellido$,nombre$,telefono$  
CLOSEIN
```

Tras estos comandos solamente se leería un cierto juego de datos; los valores de las variables podrían ser, por ejemplo:

```
apellido$    "Gonzalez"  
nombre$      "Felipe"  
telefono$    "91/3522122"
```

donde las comillas indican sencillamente el principio y final de la cadena de caracteres y no pertenecen, por tanto, a la cadena en sí. No obstante, las cadenas pueden incluir comillas.

1.5.2 Programación de ficheros secuenciales

Pero pasemos a la creación de un fichero secuencial en disco. Introduzca para ello el siguiente programa:

```
10 REM=====
20 REM      Primer programa de ficheros
30 REM=====
40 :
50 OPENOUT "test.dat"
60 PRINT #9,"Montenegro"
70 PRINT #9,"Vicente"
80 PRINT #9,"2111163"
90 REM --- Primer juego de datos ---
100 PRINT #9,"Heredia"
110 PRINT #9,"Santiago"
120 PRINT #9,"3297292"
130 REM --- Segundo juego de datos ---
140 PRINT #9,"Maldonado"
150 PRINT #9,"Mariano"
160 PRINT #9,"3419656"
170 REM --- Tercer juego de datos ---
180 CLOSEOUT
190 END
```

Hemos creado con ello un fichero con el nombre de "test.dat" que contiene tres juegos de datos.

El comando BASIC PRINT #n se maneja igual que el PRINT normal. Seguro que ya conoce el comando PRINT # en relación con las ventanas (window) que puede usted definir en su AMSTRAD. Mediante el comando PRINT # puede usted referirse a una de las ventanas definidas, con n comprendida entre 0 y 7. Con n=8 nos referimos a la impresora y con n=9 a la unidad de cassette o de disco, según la instalación. Lo mismo es válido para los comandos INPUT #n y LINE INPUT #n.

El carácter lógico de separación, carriage return se sitúa en nuestro ejemplo entre los juegos de datos, dado que el comando PRINT #9 no va seguido de punto y coma. Esto ya lo

sabia del comando PRINT usual. Si coloca un punto y coma en la última posición de la orden PRINT, la siguiente salida será escrita en la misma linea. En salidas de floppy puede usted también crear un fichero por encadenamiento de varias salidas.

Pero volvamos a leer nuestros datos:

```
10 REM=====
20 REM      Volver a leer los datos **
30 REM=====
40 :
50 OPENIN "test.dat"
60 INPUT #9,apellido$,nombre$,telefono$
70 PRINT apellido$,nombre$,telefono$
80 INPUT #9,apellido$,nombre$,telefono$
90 PRINT apellido$,nombre$,telefono$
100 INPUT #9,apellido$,nombre$,telefono$
110 PRINT apellido$,nombre$,telefono$
120 CLOSEIN
130 END
```

Obtendrá la siguiente salida:

Montenegro	Vicente	2111163
Heredia	Santiago	3297292
Maldonado	Mariano	3419656

Así hemos almacenado y recuperado con éxito datos de disco, con lo cual podemos establecer:

Mediante PRINT #9 escribimos datos en disco, y con INPUT #9 volvemos a leerlos. La entrada/salida de datos de disco no se diferencia por tanto demasiado de la entrada/salida a pantalla.

Nuestro ejemplo funciona, pero dista mucho de ser cómodo, sobre todo si se piensa en la tarea a realizar para leer 100 datos. La solución al problema es, evidentemente, un bucle, y conlleva un aligeramiento notable del trabajo:

```
10 REM =====
20 REM          Leer datos (2)
30 REM =====
40 :
50 OPENIN "test.dat"
60 FOR I=1 TO 3
70   INPUT #9,apellido$,nombre$,telefono$
80   PRINT apellido$,nombre$,telefono$
90 NEXT I
100 CLOSEIN
110 END
```

Tampoco es imprescindible que lea siempre los juegos de datos de un tirón. También puede hacer lo siguiente:

```
10 REM =====
20 REM          Leer datos (3)
30 REM =====
40 :
50 OPENIN "test.dat"
60 FOR I=1 TO 3
70   INPUT #9,apellido$
80   PRINT apellido$
90   INPUT #9,nombre$
100  PRINT nombre$
110  INPUT #9,telefono$
120  PRINT telefono$
130 NEXT I
140 CLOSEIN
150 END
```

Ha de imaginarse que, tras la apertura de un fichero, sus datos quedan disponibles. Da exactamente igual si lee un juego de datos en una única instrucción INPUT, por ejemplo

con

```
INPUT #9,apellido$,nombre$,telefono$
```

o bien si lo lee fraccionadamente, a base de varias instrucciones INPUT como por ejemplo mediante

```
INPUT #9,apellido$  
INPUT #9,nombre$  
INPUT #9,telefono$
```

Tras haber leido un juego de datos, un puntero o pointer marca el lugar donde ha de continuar la lectura.

Puede cerrar prematuramente un fichero abierto para su lectura, es decir que no ha de leer todos sus elementos.

Pero si desease leer el último elemento del fichero y realizara un nuevo intento de lectura, obtendria el mensaje de error:

EOF met

Se puede decir que el responsable de ello es un llamado puntero interno. Este puntero interno es inicializado inmediatamente tras la apertura de un fichero. Al principio (es decir, tras la apertura) este puntero señala el inicio del buffer de entrada, que se halla en la RAM del CPC, y con ello, el primer carácter en el fichero. Tras leer este carácter, el puntero se desplaza hacia la derecha y señala el siguiente carácter a leer, el cual es leido con la siguiente orden de lectura.

Leído el último carácter, este puntero interno apunta a la marca de END-OF-FILE y usted obtiene el correspondiente mensaje de error en pantalla.

El siguiente gráfico le aclarará este comportamiento del puntero interno:

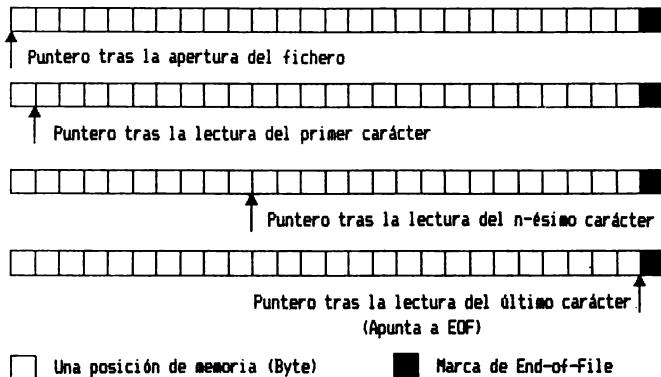


Figura 8. Puntero interno

Si la lectura se efectúa, no carácter a carácter, sino por campos de datos o incluso por juegos de datos, ello también puede representarse mediante el modelo del puntero interno. Este puntero muestra entonces, el primer juego de datos al principio, y se mueve a partir de éste hacia los siguientes juegos de datos.

La función BASIC EOF nos indica, por tanto, si en la lectura hemos alcanzado o no el último elemento de un fichero.

Para evitarlo puede utilizar la función de BASIC estándar EOF. Si no sabe exactamente cuántos juegos de datos ha de leer, resulta incluso imprescindible utilizar la función EOF. Nuestro programa ejemplo tendría el siguiente aspecto:

```
10 REM =====
20 REM      Leer datos con EOF
30 REM =====
40 :
50 OPENIN "test.dat"
60 WHILE NOT EOF
70 INPUT #9,apellido$,nombre$,telefono$
80 PRINT apellido$,nombre$,telefono$
90 WEND
100 CLOSEIN
110 END
```

Pero nuestro ejemplo es peligroso desde un principio ya que partimos de la hipótesis implícita de que nuestro fichero nos ofrecerá siempre tres campos de datos más si no hemos tropezado con EOF. Ello no es posible en ficheros de estructura desconocida, por lo que nuestro programa tendría que tomar el siguiente aspecto:

```
10 REM =====
20 REM      Lectura flexible de ficheros
30 REM =====
40 :
50 OPENIN "test.dat"
60 WHILE NOT EOF
70 INPUT #9,campo$
80 PRINT campo$
90 WEND
100 CLOSEIN
110 END
```

En este programa està descartado un error EOF-MET, ya que cada campo es leido individualmente y seguido por una prueba de EOF.

Cuando los campos de datos son leidos mediante el comando INPUT #9, son válidos como símbolos separadores de los campos de datos los siguientes caracteres:

```
Carriage Return (retorno de carro)
coma (,)
marca EOF
```

Si lee variables numèricas mediante INPUT #9, tambièn es válida la <barra espaciadora> como carácter separador (espacio).

Estos caracteres de separaciòn son fàciles de recordar dado que la coma y el retorno de carro son igualmente válidos en la rutina de INPUT normal como símbolos de separaciòn.

Por supuesto que podríamos definirnos un símbolo de fin "privado"; rellenando siempre el último campo de datos con "*fin" o algo por el estilo e ir comprobando su no apariciòn. Pero convendrà conmigo en que el manejo de la funciòn EOF es más seguro y sencillo.

Pero la mayoria de las veces, lo que deseamos no es únicamente leer los datos y darles salida directamente, sino que los datos estén en memoria para poderlos evaluar y variar. En tal caso lo mejor es utilizar un ARRAY, una variable indexada. Si esto no le suena, lea por favor el capitulo correspondiente en el manual de BASIC.

Nuestro programa de ejemplo tendría el siguiente aspecto:

```
10 DIM apellido$(3),nombre$(3),telefono$(3)
20 REM =====
30 REM      Leer-almacenar datos
40 REM =====
50 :
60 x=0
70 OPENIN "test.dat"
80 WHILE NOT EOF
90 x=x+1
100 INPUT #9,apellido$(x),nombre$(x),telefono$(x)
110 WEND
120 PRINT "Han sido leidos ";x-1;" juegos de
datos."
130 CLOSEIN
140 END
```

En la instrucción DIM de la línea 10, ha de definir el límite del índice coherentemente con el máximo del número de datos.

Mediante un bucle de este estilo se puede realizar la lectura de ficheros a través de una variable de campo, elaborarlo en el programa y, finalmente volver a almacenarlo secuencialmente. Es por tanto válida la siguiente regla de elaboración de datos, que podríamos llamar LEDS; Leer, Elaborar y Dar Salida, en este orden. Cuando la memoria lo permita, debería leer sus ficheros de este modo y gestionarlos en el programa; obtendrá enormes ventajas en cuanto a la velocidad.

1.5.3. Ficheros secuenciales y variables de campo

En programas extensos, y que tengan que gestionar una gran cantidad de datos resulta oportuno proporcionar exactamente la cantidad de datos prevista y definirla consecuentemente en el programa. A este concepto se le denomina también concepto de programa; se proporciona la llamada estructura de magnitudes.

Así que un fichero secuencial es un encadenamiento de juegos de datos, que constan a su vez de campos de datos.

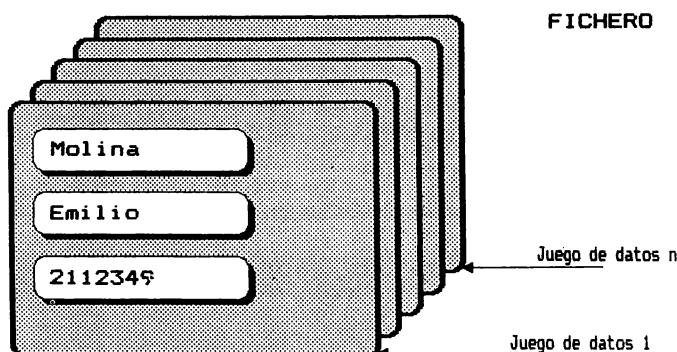


Figura 9. Fichero

Pero se puede visualizar aún mucho más el concepto de "fichero", mediante el uso de los llamados diagramas de sintaxis, usados por ejemplo en la definición de sintaxis (en la gramática, prácticamente) de lenguajes superiores de programación como el Pascal o el C. Estos diagramas de sintaxis fueron incorporados por primera vez en la definición del lenguaje estructurado de programación PASCAL de Nikolaus Wirth.

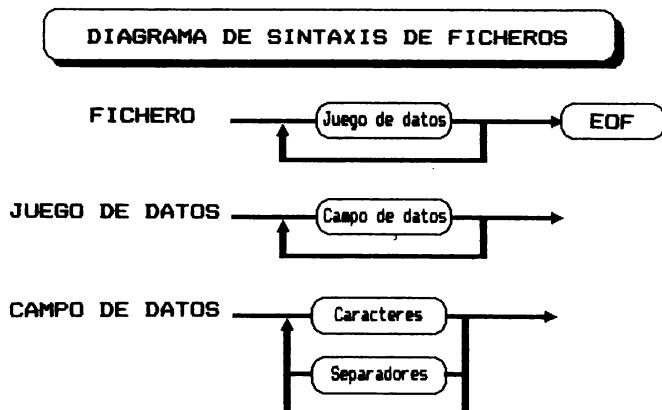


Figura 10. Diagrama de sintaxis de fichero

Si lee una única vez el fichero secuencial y dispone todos los juegos de datos en la variable de campo, ahorrará una gran cantidad de tiempo, al no tener que volver a acceder a la unidad de disco. (Un acceso a variables es más rápido aún que un acceso al disco).

Nuestras variables de campo tienen el siguiente aspecto:

INDICE	APELLIDO\$	NOMBRE\$	TELEFONO\$
1	Montenegro	Vicente	2111163
2	Heredia	Santiago	3297292
3	Maldonado	Mariano	3419656

Figura 11. Juego de datos

Así pues hemos leído estos datos una vez y los hemos almacenado en tres variables de campo unidimensionales distintas. Para un programador resulta más práctico disponer un campo bidimensional, para darle al monstruito un único nombre; ello redundaría también en beneficio de la claridad del programa.

VARIABLE DE CAMPO DE 2 DIMENSIONES

INDICE	APELLIDO	NOMBRE	TELEFONO
1	D\$(1,1)	D\$(1,2)	D\$(1,3)
2	D\$(2,1)	D\$(2,2)	D\$(2,3)
3	D\$(3,1)	D\$(2,3)	D\$(3,3)
4	D\$(4,1)	D\$(2,4)	D\$(4,3)
5	D\$(5,1)	D\$(2,5)	D\$(5,3)

Figura 12. Campo bidimensional

Nuestra variable de campo D\$(x,y) consta de 5 juegos de datos con 3 variables de campo cada uno. Esta variable de campo hubo de ser "dimensionada", es decir, creada mediante la instrucción

```
DIM D$(5,3)
```

La variable de campo D\$(x,y) contendría, por ejemplo, las siguientes anotaciones:

D\$(x,1)=apellido	antes apellido\$(x)
D\$(x,2)=nombre	antes nombre\$(x)
D\$(x,3)=telefono	antes telefono\$(x)

Nuestro ejemplo tendría el siguiente aspecto:

```
10 REM =====
20 REM      Leer en D$(x,y)
30 REM =====
40 DIM D$(3,3)
50 REM  3 juegos & 3 campos de datos
60 OPENIN "test.dat"
80 FOR I=1 TO 3
90   FOR J=1 TO 3
100  INPUT #9,D$(I,J)
110  NEXT J
120 NEXT I
130 :
140 REM =====
150 REM      SALIDA
160 REM =====
170 FOR I=1 TO 3
180   FOR J=1 TO 3
190     PRINT D$(I,J),
200   NEXT J
210 PRINT
220 NEXT I
230 CLOSEIN
240 END
```

Puede tomar este programa de ejemplo como muestra cuando escriba un programa de ficheros. Mediante el anidamiento de dos bucles conseguirá una lectura en secuencia lógica de los campos y juegos de datos. Si desea modificar datos en un programa de ficheros, es mejor que lea todos los datos mediante una variable de campo de este estilo. Después puede elaborar estos datos, corregirlos, etc... Al fin del programa, los datos han de volver a ser almacenados bajo el mismo nombre.

Generalmente, se crea además un fichero auxiliar, en el que se encuentran informaciones sobre la estructura de magnitudes. Por dicha estructura entendemos la cantidad de datos anotados en el fichero y cuántos campos contiene cada

juego de datos. Si crea un programa flexible de ficheros, serà ello imprescindible por motivos prácticos. Además, de este modo puede ahorrar espacio de memoria, lo cual siempre resulta importante, ya que cada coordenada innecesaria cuesta memoria y tiempo. El dimensionado en la instrucción DIM puede ser dinàmico, es decir, puede usted incluir variables como límites de índice. Ya crearemos un fichero de informaciones de este tipo. Para ello tendremos que incorporar el número de JUEGOS DE DATOS y el número de CAMPOS DE DATOS por juego.

```
OPENOUT "test.inf"
WRITE #9,3,3
CLOSEOUT
```

Pondremos "INF" a este tipo de fichero por fichero de INformaciones. El nombre de fichero ha de ser en nuestro ejemplo igual al nombre del fichero(.dat). He aquí la rutina flexible de lectura:

```
10 REM =====
20 REM   Rutina flexible lectura
30 REM =====
40 :
50 INPUT "nombre de fichero : ";file$
60 OPENIN file$+".inf"
70   INPUT #9,juegos,campos
80 CLOSEIN
90 :
100 DIM d$(juegos,campos)
110 :
120 OPENIN=file$+".dat"
130 FOR I=1 TO juegos
140   FOR J=1 TO campos
150     INPUT #9,d$(I,J)
160   NEXT J
```

```
170 NEXT I  
180 CLOSEIN  
190 END
```

Ejecute el programa e introduzca "test".

En la linea 100 se dimensionará, según las necesidades. En los casos en los que el número de juegos de datos pueda aumentar durante la ejecución del programa, ello ya ha de ser previsto al dimensionar. Otra solución es la de fijar previamente el número máximo de juegos de datos; ello no impide que el número de campos de datos siga siendo dinámico, ya que lo más probable es que éste no cambie. Pero la lectura del número de juegos de datos y de campos de datos sigue siendo una ayuda estimable. Si cambia la linea 100 por la linea

```
100 DIM d$(200,campos)
```

conseguirá que el número de campos dependa del fichero.

Hasta ahora, en la apertura de ficheros sólo hemos considerado nombres de fichero fijos. Lo que es lo mismo, el nombre de fichero no es variable, se halla entre comillas y es fijado en la programación. De hecho, a veces se abren ficheros mediante variables de cadena, tal y como hicimos en nuestro ejemplo (lineas 60 y 120). Lamentablemente, en este aspecto el sistema operativo aún no está suficientemente maduro. El AMSDOS abre en ocasiones el fichero bajo un nombre equivocado o bien reacciona mediante un mensaje de error. Si, tras este mensaje de error, da salida a la variable de cadena correspondiente, reconocerá con dificultad que el contenido de la variable es correcto.

Al abrir un fichero, en la RAM del Amstrad se prepara un buffer de 4096 bytes. Esta memoria intermedia es libremente desplazable y se usa para almacenar de modo temporal los datos que son leídos o bien han de ser escritos en disco. Esta memoria intermedia suele estar en la región superior de

la memoria. El puntero para la posición de memoria más alta disponible para el BASIC (HIMEM) es reducido en ello exactamente en 4096 Bytes, y esto hasta que el fichero abierto vuelva a ser cerrado. El buffer del floppy se halla entonces por encima de este puntero de fin de memoria HIMEM (ver también figura).

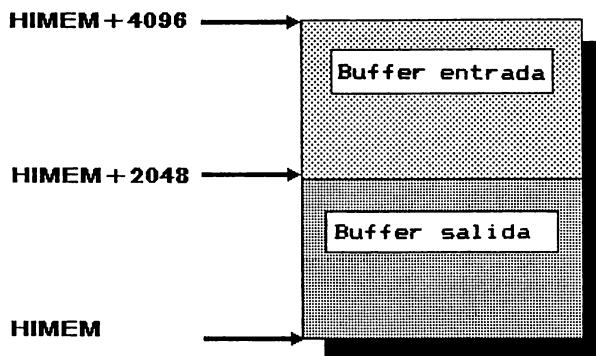


Figura 13. Buffer de floppy

Pero por esta región superior se agrupan las variables de cadena. Aquí pueden darse, con facilidad, "colisiones", en especial al trabajar con variables de campo, dado que entonces el número de cadenas es especialmente elevado.

De tanto en tanto, se hace necesario "hacer limpieza" de la zona de memoria asignada a las variables de cadena. De modo distinto a lo que ocurre con las variables numéricas, ya que de las manipulaciones de cadenas quedan trozos de cadena en la memoria. Se denomina garbage collection a esta "limpieza"; es una especie de reorganización de la memoria intermedia de cadenas. En ello puede ocurrir que este buffer de diskette se convierta en la víctima de la reorganización. Si el buffer es desplazado por la garbage collection, el nombre de fichero no puede volver a ser leído satisfactoriamente. Con esta corta rutina podemos evitarlo.

```
OPENOUT "Dummy"  
MEMORY HIMEM-1  
CLOSEOUT
```

Esta secuencia de instrucciones debería incorporarse al comienzo de cada programa que trabaje con el diskette. Se reserva cuidadosamente el buffer de 4096 bytes, y es protegido mediante el comando MEMORY (el puntero de HIMEM se desplaza coherentemente). Ha de usar este comando antes de que se disponga la primera variable de cadena. A partir de ese momento, el buffer de diskette ya no puede ser destruido por la garbage collection, ya que el buffer está protegido de ésta por el comando MEMORY.

El nombre "Dummy" en la instrucción OPENOUT no indica, en este caso, fichero alguno (no se escribe en él), sino que es un término del argot para una especie de "cubo electrónico de basura". Antes bien, mediante el comando OPENOUT "Dummy" se consigue la creación del antes mencionado buffer de 4096 bytes. La variable del sistema HIMEM es consecuentemente decrementada para preservar de variables al buffer. Tras un CLOSEOUT, HIMEM volvería a ser desactivada, para evitarlo se utiliza el comando MEMORY.

Así que, en caso de que obtuviera un mensaje de error en nuestro programa de ejemplo, que no contiene esta rutina, incorpore la siguiente línea 40:

```
40 OPENOUT "Dummy" : MEMORY HIMEM-1 : CLOSEOUT
```

Queda aún otro peligro, consistente en que el usuario entre un nombre de fichero incorrecto, lo que llevaría a la ruptura del programa. Pero también esto puede ser evitado creando por ejemplo un fichero en el que estén registrados todos los nombres de fichero - en la práctica, pues, un "directorio privado". Llamaremos a este fichero "filename.dir".

```
OPENOUT "filename.dir"
PRINT #9,"test"
CLOSEOUT
```

Nuestro directorio privado sólo está disponible a partir de la entrada "test". Ya sólo hemos de construirnos esta rutina de test, y quedará descartado un mensaje de error.

```
10 REM =====
20 REM Rutina flexible de lectura
30 REM =====
40 OPENOUT "Dummy" : MEMORY HIMEM-1 : CLOSEOUT
50 INPUT "Nombre de fichero : ";file$
60 GOSUB 1000
70 IF hallado=0 THEN 50
80 OPENIN file$+".inf"
90   INPUT #9,juegos,campos
100 CLOSEIN
110 :
120 DIM d$(200,campos)
130 :
140 OPENIN=file$+".dat"
150 FOR I=1 TO juegos
160   FOR J=1 TO campos
170     INPUT #9,d$(I,J)
180   NEXT J
190 NEXT I
200 CLOSEIN
210 END
2100 REM =====
2101 REM Nombre permitido ?
2102 REM =====
21030 OPENIN "filename.dir"
21040 hallado=0
21050 WHILE NOT EOF AND hallado=0
21060   INPUT #9 apellido$
21070   IF apellido$=file$ THEN hallado=1
```

```
1080 WEND  
1090 CLOSEIN  
1100 RETURN
```

Una comprobaciòn como ésta, o similar, resulta inexcusable para estructurar un programa de forma que resulte de uso agradable. Las entradas erròneas deberian ser detectadas y bloqueadas, en la medida de lo posible.

En cualquier momento, sólo es válido un canal de entrada abierto y un canal de salida abierto. De modo que puede abrir como màximo dos canales hacia el floppy. Los canales son los intermediarios entre el sistema operativo del CPC y el AMSDOS, que es el encargado del almacenamiento y la lectura de datos en disco. Existen en el CPC ademàs, y a titulo de ejemplo, un canal a la pantalla y un canal al teclado. Cada uno de estos canales dispone de un nùmero lògico, mediante el cual puede uno referirse a los distintos aparatos.

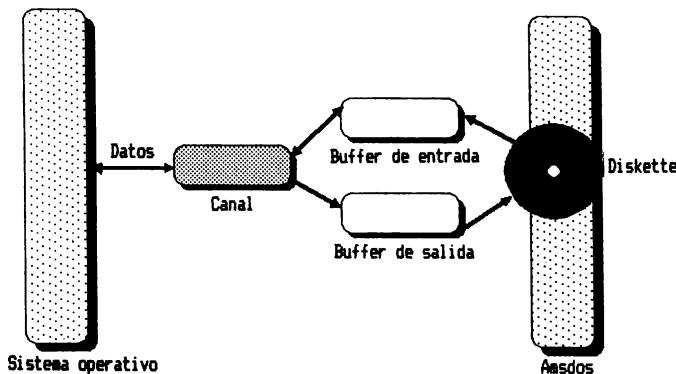


Figura 14. Canales & buffer

En la subrutina que comienza en la linea 1030 se comprueba si existe un fichero con el nombre file\$. Para ello se comparan todos los apuntes existentes en el fichero "filename.dir" con la variable file\$. En caso de que la rutina llegue al final del fichero sin haber encontrado el nombre, la variable "hallado" es igual a cero (0). La variable es puesta a 1 en caso de que la comparación se verifique. Este valor puede ser comparado en el programa principal, pudiéndose actuar en consecuencia.

Si desea por ejemplo escribir un cómodo programa de gestión de direcciones, sería aconsejable incluir en el mismo rutinas de protección de este estilo

Naturalmente, hemos de asegurarnos previamente de que el fichero "filename.dir" exista, ya que en caso contrario aparecería el mensaje File not found. Para evitar esta circunstancia podríamos incorporar un punto en el menú llamado ''Reinicialización del sistema'', o algo por el estilo, que crearía este fichero.

Todos estos 'trucos' están incluidos en el programa de gestión de ficheros que hallará en el capítulo 5 de este libro. Se recomienda trabajar los diferentes apartados del programa para comprender el manejo de las rutinas.

1.5.4 Diferencias entre PRINT # y WRITE

Hasta ahora hemos usado únicamente el comando PRINT. Seguramente ya habrás observado, tanto en el manual de programación como en el presente libro, que también existe el comando WRITE #.

En nuestro programa ejemplo se utilizó carriage return como carácter separador. Pero puede recordar que carriage return no es el único carácter separador, la coma (,) también sirve como tal, tanto para cadenas como para datos numéricos. En el caso de datos numéricos, a uno le puede resultar indiferente el que vayan separados por una coma, un carácter separador o simplemente un espacio. Pero en el caso de variables de cadena, a uno le pueden empezar a suceder cosas desagradables. Pruebe el siguiente ejemplo:

```
10 REM =====
20 REM      Ejemplo de PRINT #
30 REM =====
40 :
50 OPENOUT "Demo"
60 PRINT "9,"Gimenez, Juan"
70 PRINT #9,"Espasa, Susana"
80 CLOSEOUT
90 :
100 REM=====
110 REM      Nueva lectura
120 REM=====
130 :
140 OPENIN "Demo"
150 INPUT #9, persona1$
160 INPUT #9, persona2$
170 CLOSEIN
180 PRINT "persona1 = "persona1$
190 PRINT "persona2 = "persona2$
200 END
```

Seguro que espera que la salida a pantalla sea algo parecido a esto:

```
personal1 = Gimenez, Juan
persona2 = Espasa, Susana
```

Algunos de ustedes ya lo sabrán: por lamentable que sea, el resultado no tiene este aspecto. Cuando ejecute el programa se llevará una sorpresa:

```
personal1 = Gimenez
persona2 = Juan
```

En el ejemplo se ve claramente que la coma funciona como carácter separador y que en la instrucción INPUT #9 ha conducido a la separación de las dos cadenas. Faltan también la coma en la cadena de caracteres y el carácter vacío delante de "Juan". La solución del problema es muy sencilla: al utilizar el comando WRITE # la cadena a imprimir será puesta entre comillas. Al leer esta cadena, todos los espacios vacíos y comas serán transferidos a la cadena. Únicamente las comillas han de dejar de ser consideradas de dicha cadena, como efectivamente ocurre.

Mediante el comando WRITE puede también dar salida a pantalla a valores y aclarar así el funcionamiento del comando. Introduzca:

```
WRITE 1,2,"Si, si, esto es así.",3
```

En la pantalla aparecerá:

```
1,2,"Si, si, esto es así.",3
```

Si hace lo mismo mediante la instrucción PRINT, aparecerá lo siguiente en la pantalla:

1 2 Si, si, esto es así. 3

Una coma en la instrucción PRINT provoca el salto del tabulador a la siguiente posición de tabulado, lo que produce grandes espacios vacíos. En el disco se escribe igual que en pantalla, es decir, se transferirían igualmente una gran cantidad de espacios (lo que, por cierto, no ahorra memoria que digamos).

Lo que es fácil constatar es que, a causa de las comas, la cadena ya no puede ser reconocida como tal. En el comando WRITE, con las comillas no cabe confusión respecto al principio y final de la cadena.

Como agradable efecto secundario obtenemos el ahorro de espacio en el disco, si se desean almacenar datos numéricos.

También resultaría posible simular el comando WRITE mediante el comando PRINT:

```
PRINT 1",";2","CHR$(34);"Si, si, esto  
es así.";CHR$(34);";3
```

El resultado sería ahora el mismo en la pantalla y en el disco, además de conseguir en la lectura el efecto deseado. Pero el comando WRITE es más sencillo y cómodo, por ello debería ser usado más a menudo. Resulta especialmente práctico cuando se desea escribir varios campos de datos en disco en una linea BASIC. Como se desprende del programa, con cada linea PRINT # hemos grabado en el disco un único campo de datos, enviando con ello un carriage return como símbolo separador, ya que no tenemos punto y coma alguno al final de la instrucción PRINT #. Así que con la instrucción

WRITE nuestro programa se hace más corto y claro, presentando el siguiente aspecto:

```
10 REM =====
20 REM      Ejemplo de WRITE #
30 REM =====
40 :
50 OPENOUT "Demo"
60   WRITE #9,"Gimenez, Juan"
70   WRITE #9,"Espasa, Susana"
80 CLOSEOUT
90 :
100 REM=====
110 REM      Nueva lectura
120 REM=====
130 :
140 OPENIN "Demo"
150   INPUT #9,persona1$
160   INPUT #9,persona2$
170 CLOSEIN
180 PRINT "persona1 = "persona1$
190 PRINT "persona2 = "persona2$
200 END
```

Tras ejecutar el programa observarà que hemos llegado al resultado deseado.

Así que, tras una instrucción PRINT # y una instrucción WRITE #, automáticamente se escribe un carácter separador, el carriage return. Pero hay casos, en los que se desea renunciar a ello, por ejemplo al calcular cadenas de caracteres e ir dàndoles salida una detrás de otra. Ahora nos proponemos dar salida a todo el alfabeto, y no solucionamos el problema mediante una simple instrucción PRINT #, sino que programamos un bucle.

```
10 REM =====
20 REM Ejemplo de encadenamiento
30 REM =====
40 :
50 OPENOUT "Demo"
60 FOR i=1 TO 26
70   PRINT #9,CHR$(64+i);
80 NEXT i
90 PRINT #9
100 CLOSEOUT
110 OPENIN "Demo"
120 INPUT #9,a$
130 PRINT a$
140 CLOSEIN
150 END
```

En la linea 60 definimos un bucle de indice i, que va de 1 a 26. En efecto, el alfabeto (de nuestro Amstrad) tiene 26 letras. El código ASCII de "A" es 65, de modo que podemos obtener en la linea 70 el código ASCII del carácter imprimible a partir del valor actual del indice del bucle sumándole 64. (Código ASCII proviene de American Standard Code for Interchange y es utilizado en la mayoría de ordenadores domésticos y personales para la codificación del alfabeto y caracteres especiales). Lo importante es que el último carácter en la linea 70 es un punto y coma. Al igual que en el BASIC normal, al efectuar una salida a pantalla, el punto y coma tiene en el disco el efecto de evitar que se envie un "fin de campo de datos", comparable al 'envío de un 'fin de linea' en la pantalla.

Tras iniciar el programa, vemos que la variable a\$ contiene el alfabeto completo. A buen seguro ha quedado aclarado el significado del punto y coma. En este ejemplo en concreto, no hubiera podido utilizar de ningún modo el comando WRITE. La salida hubiera tenido el siguiente aspecto:

```
"A""B"C""D""E""F""G""H""....."Y""Z"*
```

mientras que con nuestro ejemplo obtenemos el siguiente resultado:

```
ABCDEFGH...YZ*
```

(* representa de nuevo carriage return)

Como ve, para cada caso ha de reflexionarse acerca de cuál es el comando más indicado. A menudo resulta indiferente el usar PRINT #9 o WRITE #9. A veces WRITE #9 es más efectivo, pero en ocasiones, como en este ejemplo, es totalmente inutilizable.

Intente deducir el valor que contendría la variable a\$ si sustituysésemos la linea 70 por:

```
70 WRITE #9,CHR$(i+64);
```

Si ha llegado a una conclusión, cambie la linea y ejecute el programa. ¿Acertó? Entonces puede pasar al capítulo próximo.

1.5.5 Diferencias entre INPUT # y LINE INPUT

Ya habrá comprendido las diferencias entre PRINT #9 y WRITE #9. También para la lectura de datos existen dos órdenes distintas: la orden INPUT #9 normal, que hemos utilizado hasta ahora, y la orden LINE INPUT #9. También entre ellas existen diferencias sustanciales.

Nos limitaremos en nuestra comparación entre INPUT y LINE INPUT a variables de cadena, dado que el sentido de la instrucción LINE INPUT sólo queda claro al manejar variables de cadena.

Sabemos que los campos de datos están separados entre sí por símbolos separadores. Estos símbolos separadores pueden ser el retorno de carro, la coma y el carácter EOF. Por lo tanto, no podremos leer mediante INPUT #9 una cadena que contenga una coma (a menos que encerremos la cadena entre comillas), ya que la coma sería interpretada automáticamente como un símbolo separador. De ello surge otro problema: si las comillas marcan una cadena, ¿cómo puedo leer comillas? Las comillas no pueden en absoluto ser leídas de una cadena mediante INPUT, esto está claro.

Aquí es donde aparece el comando LINE INPUT. LINE INPUT lee cualquier carácter; solamente el retorno de carro es válido como símbolo separador, por lo que las comas y comillas son aceptadas por LINE INPUT.

Si utiliza el comando

```
WRITE #9,1,2,"Si, si, esto es así.",3
```

para escribir en disco, y vuelve a leerlo con

```
LINE INPUT #9,todo$
```

la variable todo\$ tendrá el siguiente contenido:

```
1,2,"Si, si, esto es así.",3
```

Es decir, se incluirían todos los símbolos separadores en la cadena sin excepción, incluidas las comillas.

El comando LINE INPUT no siempre resulta mejor que el INPUT; de hecho, el cambiar simplemente todos los INPUT #9 de un programa por LINE INPUT #9 tendría, en general consecuencias catastróficas. Ya se puede imaginar que el motivo de ello gira alrededor del hecho de que son leídos los separadores. Si al utilizar los comandos PRINT o WRITE no se considera

que la lectura se efectuará mediante LINE INPUT, con lo cual solamente el retorno de carro continuará separando, el resultado puede parecer cosa de brujas. Si escribe un programa propio que acceda frecuentemente a disco, piense bien qué órdenes utilizar para la escritura y la lectura.

Pero pongamos un ejemplo en el que la orden LINE INPUT resulta indispensable, dado que ha de ser leído un fichero de contenido y estructura desconocidos.

Cargue uno de los programas de este libro y almacénelo directamente mediante

```
OPENOUT "ASCII.DAT"  
LIST #9  
CLOSEOUT
```

Ha almacenado en disco el listado del programa bajo el nombre "ASCII.DAT". No resulta posible abrir un fichero de programa normal para su lectura mediante la orden OPENIN, dado que este fichero tiene un header (=cabecera), que no es aceptado con la instrucción OPENIN. Así que tras haber LISTado un fichero de programa en disco, introduzca el siguiente programa:

```
NEW  
  
10 REM ======  
20 REM          Lectura de cualquier fichero  
30 REM ======  
40 :  
50 OPENIN "ASCII.DAT"  
60 WHILE NOT EOF  
70 LINE INPUT #9,linea$  
80 PRINT linea$  
90 WEND
```

```
100 CLOSEIN
110 END
```

Esta rutina le mostrará en la pantalla el listado de su programa, con todas las comas y comillas, como en el programa original. Para tareas como ésta o similares resulta muy indicado el comando LINE INPUT que recoge del disco todos los caracteres sin ningún problema.

Lamentablemente, en el BASIC del Amstrad no está prevista la lectura de caracteres individuales del fichero abierto para lectura, al contrario de otras versiones de BASIC para otros ordenadores. Esta posibilidad tendría, obviamente, la virtud de poder hacer legible incluso el carácter de retorno de carro. Sin embargo, existe una rutina en el DOS, DISC IN CHAR, que puede usted utilizar para ello (ver listado DOS) - de hecho, sólo en lenguaje máquina. Puede usted entonces escribir para su problema específico una rutina que transfiera el carácter leído a su programa BASIC.

Aquí tiene una corta rutina que simula la orden BASIC GET. Cuando salte a la rutina, el fichero ha de estar abierto para la lectura.

```
10000 REM =====
10010 REM           Rutina G E T
10020 REM =====
10030 :
10040 IF LEN(in$)=0 THEN 10080
10050 ch$ = LEFT$(in$,1)
10060 in$ = MID$(in$,2)
10070 RETURN
10080 IF EOF THEN CLOSEIN : ch$="" : RETURN
10090 LINE INPUT #9,in$
10100 IF NOT EOF THEN in$ = in$ + CHR$(13)
10110 GOTO 10050
```

El carácter leído será transferido a la variable ch\$, siendo el retorno de carro un carácter lícito. Tras leer el último carácter, ch\$ estará vacía, por lo que LEN(ch\$) será igual a cero.

Para leer en diálogo por pantalla valores numéricos (cifras) usted suele utilizar el comando

```
10 INPUT valor
```

En caso de que introduzca un valor no numérico, el sistema se manifiesta mediante

```
?Redo from start
```

Naturalmente, el AMSDOS no puede dar salida a un mensaje de este tipo. Algunos sistemas envían un FILE-TYPE-ERROR, lo que viene a significar: se transfirió una cadena en lugar de un valor numérico. El AMSDOS no envía mensaje de error alguno, sino que lleva a cabo, internamente, el siguiente cálculo:

```
10 INPUT #9,a$ : valor=val(a$)
```

Aquí queda claro porqué el espacio vacío sirve de carácter separador en el caso de cifras. Aparte de éste, cualquier otro carácter no numérico sirve de separador, con la excepción de "E" o bien de "e", preciso para la representación en formato exponencial.

1.6 Almacenamiento relativo de datos

1.6.1 ¿Qué es el almacenamiento relativo de datos?

Como ya mencionamos anteriormente, todas las técnicas de almacenamiento de datos se reducen en esencia al almacenamiento secuencial o bien al almacenamiento relativo (de datos). Ya hemos tratado en profundidad el almacenamiento secuencial, queremos ahora saber lo que se ha de entender por almacenamiento relativo de datos.

En ello se plantea para usted, como propietario de un CPC, un problema inmediato. En el sistema operativo del Amstrad no se prevé, lamentablemente, el almacenamiento relativo de datos; ello significa que no existen órdenes, de modo que la explicación del presente tema a base de ejemplos puede presentar dificultades. Pero solventaremos en común también este problema cuando hayamos tratado la parte teórica del almacenamiento relativo de datos.

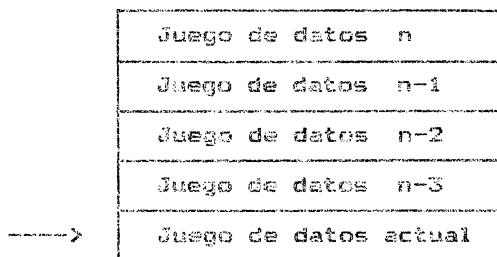
En los ficheros secuenciales podíamos abrir un fichero, bien para la lectura, o bien para la escritura. En caso de que el fichero sea abierto para la escritura, el contenido anterior del fichero resulta totalmente sobreescrito; en la práctica, si dispone el fichero bajo el mismo nombre, pero sin que el contenido anterior juegue papel alguno. Ciento que el sistema operativo del CPC crea una copia de seguridad, el fichero BAK, pero ello supone tan sólo un consuelo mínimo.

Un sistemas operativos algo más cómodos, se pueden encadenar datos a un fichero secuencial ya existente. Es decir, que un fichero no se abre para escritura, con lo cual quedaría destruido, sino para el encadenamiento. El puntero interno apunta tras la apertura directamente al último elemento. Las siguientes acciones de escritura serán encadenadas a los datos existentes. En inglés se denomina a esto Append. Así que el contenido de un fichero se conserva y se encadenan únicamente datos nuevos al final de este fichero existente.

Ello puede ser, por ejemplo, interesante en un (fichero) diario para contabilidad.

Su aspecto es totalmente distinto en la gestión relativa de datos. En ella, un fichero se abre siempre para la lectura y la escritura simultánea. Hemos de olvidarnos de que un carácter (byte) ha de seguir obligatoriamente al otro. De modo distinto a lo que ocurre en el almacenamiento secuencial de datos, en el almacenamiento relativo de datos se puede acceder libremente a cualquier carácter del fichero.

Nos hemos imaginado el fichero secuencial como una caja de fichas, en la que cada ficha está detrás de otra. En ello, cada ficha representa un juego de datos.



Si queremos llegar al n -ésimo juego de datos de nuestro ejemplo, todos los juegos de datos antecedentes han de ser sobreleídos. Un proceso con gran consumo de tiempo. Si volvemos a nuestro modelo de la caja de fichas, deberíamos leernos cada ficha que precede a la n -ésima.

Pero con el almacenamiento relativo de datos podría usted señalar directamente, mediante el puntero, el juego de datos n y leer su contenido, sin preocuparse de leer el contenido de los tres juegos de datos que de hecho le anteceden; el

puntero interno es, por tanto, movable a voluntad. La siguiente figura aclarará esta movilidad.

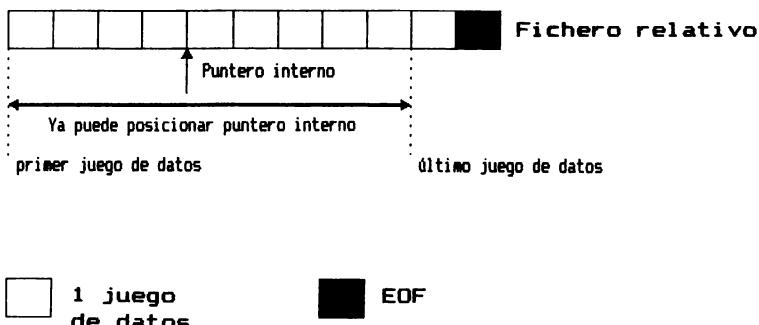


Figura 15. Puntero interno

Si disponemos de un fichero relativo, ello se puede comparar, en nuestro modelo de la caja de fichas, a poder elegir la ficha deseada.

El almacenamiento relativo de datos se denomina en inglés Random Acces, lo que viene a significar acceso directo o aleatorio. También se puede aclarar de modo "relativamente" sencillo el término "relativo". El acceso a un juego de datos se efectúa siempre en relación al inicio del fichero.

Imaginese que abre un cajón, en el que hay 100 carpetas. Cada carpeta representa un juego de datos, con abundante información acerca de un cliente, por ejemplo. Puede extraer

la carpeta que deseé, y reconoceré la deseada, por ejemplo, por un título escrito sobre ella.

Pero el almacenamiento relativo de datos no tiene sólo ventajas. Si bien el acceso y el manejo resultan mucho más cómodos que en el almacenamiento secuencial de datos, resulta imprescindible hacerse una idea de la magnitud de la estructura antes de la creación de un fichero relativo. En ello cuentan tanto la magnitud de los juegos de datos como el número previsible de juegos de datos que ha de comprender el fichero. Y es que, de modo distinto a lo que ocurría con el fichero secuencial, para los ficheros relativos hay que disponer previamente un fichero, antes de que pueda ser escrito. También aquí hay diferencias: en algunos sistemas operativos pueden luego encadenarse más juegos de datos - en otros sistemas no se da esta posibilidad. Aceptaremos que no pueden encadernarse juegos de datos, y ello por dos motivos:

- a) en "nuestra" gestión relativa de ficheros ello no resulta posible, ya que no fue previsto,
- b) uno debería acostumbrarse a reflexionar siempre acerca de la magnitud y esquema de un fichero, ya que se reducen los errores y no surgen desagradables sorpresas.

1.6.2 La disposición de un fichero relativo

Pero, ¿por qué he de hacerme una idea sobre el tamaño de cada juego de datos en la creación de un fichero relativo?, se preguntará.

Ya comentamos que el almacenamiento relativo de datos sólo resulta posible, a causa del sistema operativo, si se determina con exactitud la longitud de cada juego de datos.

Pero las limitaciones no son tan intratables como quizás piense usted en este momento.

Usted puede, evidentemente, no alcanzar esta longitud máxima de juego de datos definida, de modo que lo que realmente fija es un espacio máximo de juego de datos. De hecho, debería vigilar, en su programa, que no se sobrepase esta longitud máxima. Si ello sucediese, el juego de datos siguiente sería sobreescrito y con ello parcial o totalmente borrado, con una probabilidad casi absoluta.

También depende de la máquina en la que programe el que tenga que definir simplemente la longitud de juego de datos o bien la longitud de cada campo de datos de forma explícita. En el PC de IBM, por ejemplo, ello es de tal modo que ha de definirse la longitud de cada campo de datos. Una característica muy razonable del potente MS-BASIC, ya que de esta forma no se pueden sobreponer las estrictas fronteras de los campos de datos, lo que conduce a un disciplinado estilo de programación.

En "nuestra" gestión relativa de ficheros (realizada en el capítulo 5) la situación es la siguiente: la longitud de los campos de datos a crear puede variar, pero sin que la suma de las longitudes de los campos de datos sobreponga la longitud máxima de juego de datos.

Pero pasemos a ver de un modo más detallado cómo se calcula la tantas veces citada longitud máxima de juego de datos.

Un juego de datos -como vimos en el capítulo 1.5- consta de uno o más campos de datos. Los campos de datos que se precisan depende obviamente muchísimo de la situación del momento. En nuestro ejemplo queremos referirnos al tratamiento convencional de datos, por lo que tomaremos como ejemplo un sencillo juego de datos de direcciones. Este juego de datos ha de contener los siguientes campos de datos:

Nombre (y apellidos)	35 caracteres
Calle (+ número)	20 caracteres
Población + provincia	22 caracteres
Teléfono	15 caracteres
<hr/> TOTAL:	92 caracteres

Así que cada juego de datos comprende 92 caracteres. Si programamos con un sistema que permita flexibilidad longitudes de campos de datos, hemos aún de añadir 4 caracteres a estos 92 calculados. Precisamos estos 4 caracteres como caracteres separadores de los distintos campos de datos. La gestión relativa de ficheros incluida en el presente libro también necesita estos 4 caracteres separadores - incluya, por tanto, los caracteres separadores en su cálculo. Llegamos así a una longitud total de juego de datos de:

96 caracteres

En caso de que un juego de datos no alcance esta longitud, el espacio restante para este juego de datos será "rellenado" en el disco.

Tras haber calculado la longitud de nuestro juego de datos, deberíamos pensar en cuántos de estos juegos de datos precisamos. Como ya indicamos, hay sistemas en los que se pueden encadenar juegos de datos con posterioridad - pero ello no es la regla, y resulta imposible en nuestra gestión relativa de datos. Es por ello que deberá estimarse con la máxima precisión posible la cantidad de juegos de datos que se espera en un futuro, ya que tal cantidad será la que tenga que ser dispuesta.

Supongamos que usted tiene 60 fichas de este estilo, que desea gestionar por ordenador. Usted espera un incremento de

un 50 %. En tal caso, no deberia ahorrar, y deberia prever y disponer tranquilamente 120 o incluso hasta 150 juegos de datos.

En el capitulo 5 le aclararemos còmo disponer estos juegos de datos, precisamente cuando le expliquemos los nuevos comandos.

1.6.3 ¿Còmo trabajar con un fichero relativo?

Una vez creado en el disco espacio suficiente para recibir la cantidad prevista de juegos de caracteres, se habrà conseguido lo más importante para disponer de este fichero relativo. De modo distinto a lo que ocurre con los ficheros secuenciales, podemos renunciar a leer todo el fichero en memoria - por ejemplo, en una variable de campo -. Ya que el manejo del almacenamiento relativo de ficheros posibilita el acceso directo, ràpido y sin problemas, se puede hablar, refiriéndonos al fichero relativo, casi como de una variable de campo externa.

Esta comparaciòn no està tomada por los pelos. Hemos dispuesto en el disco el fichero relativo. Ello es comparable al dimensionamiento de una variable. Mediante el comando DIM se reserva espacio en memoria para la variable.

De modo igualmente sencillo al del caso de una variable de campo puede usted tambièn acceder en el fichero relativo a cada componente existente. En el argot se denomina a esta "componente", que representa un juego de datos, RECORD o registro. Así que cada juego de datos (record) tiene un nùmero de record fijo asignado que lo identifica. Se plantea ahora la cuestiòn de si hemos de numerar nuestros 150 juegos de datos de 0 a 149, o bien de 1 a 150. En la mayoria de

los equipos se comienza por el número de record 0, de modo que no hace usted nada incorrecto si comienza por el número de record 1, a lo sumo desaprovecha un record.

Un pequeño consejo: Incluso en una memoria externa, ha de anotar en alguna parte cuántos records están ocupados en su fichero. Puede disponer, evidentemente, estos datos y otros similares en un fichero secuencial, pero la práctica indica que resulta mucho más útil "desaprovechar" el record 0 exclusivamente para estos fines. De este modo, además, no le falta a uno la referencia entre un fichero secuencial y otro relativo. Pero no olvide, con vistas a la seguridad, incluir un juego de datos más en su cálculo.

Como ya mencionamos, un fichero relativo siempre es simultáneamente abierto para la lectura y la escritura. Dado que lo más frecuente es el acceso a un fichero relativo, y en muy repetidas ocasiones, se suele abrir al principio, y se cierra poco antes de finalizar el programa. De este modo se ahorra mucho tiempo que se consumiría en las aperturas y los cierres. Lamentablemente, en el CPC no resulta posible mantener abiertos simultáneamente un fichero relativo y uno secuencial. La causa de ello es que el espacio para el buffer de datos con sólo dos ficheros abiertos resulta ya bastante pobre en el CPC - y esto no lo podemos cambiar. Dado que el fichero relativo es abierto para lectura y escritura simultáneamente, se ocupan ambos buffers en el CPC.

Una vez abierto un fichero relativo, el acceso a un record cuaquiera deja de ser un problema. El número de record es suficiente para la identificación. Por regla general, el número de identificación se da en el comando de lectura o escritura, por ejemplo:

```
GET #1,11,A$
```

En PASCAL se transferirà el nùmero de record como primer paràmetro del comando WRITE. Tambièn se da el caso de que se elija el record deseado mediante un comando correcto, como por ejemplo, SEEK o bien RECORD. Es este el caso de nuestra gestiòn relativa de ficheros: antes de poder leer o escribir un juego de datos, éste se elige mediante el comando RECORD. Ello permite un elegante paso a la pregunta:

1.6.4 ¿Còmo funciona la gestiòn relativa de ficheros?

Hasta ahora no hemos mencionado el motivo por el que el sistema operativo precisa la longitud de record. Pero es que sin longitud de record, no se puede hacer nada.

Hemos comentado, que el sistema operativo rellena el espacio restante ante juegos de datos de longitud menor que la longitud màxima de juego de datos. Pero el sistema operativo sólo puede hacerlo si conoce esta longitud màxima. Esta es la primera, pero no la más importante razòn por la cual el sistema operativo ha de conocer esta longitud de record.

Mucho mès importante aun resulta esta informaciòn en la determinaciòn de la posiciòn relativa respecto al inicio del fichero a partir del nùmero de record. Un fichero con 150 juegos de datos de 96 caracteres cada uno no es otra cosa que una sucesiòn de

$$150 * 96 = 1440 \text{ caracteres.}$$

Si desea leer, por ejemplo, el juego de datos de nùmero de record 77, éste se halla, en relaciòn al inicio del fichero, en la posiciòn:

$$77 * 96 = 7392$$

Mediante este número, 7392, que le proporcionamos, el sistema operativo sabe qué puede pasar por alto en la lectura 7392 caracteres. Con una magnitud de sector de 512 bytes, el juego de datos número 77 se hallará, por tanto,

$$7392 / 512 = 14,43375$$

en el sector 14 del fichero relativo. Observe que mediante este proceso de cálculo existe también el sector cero. En el directorio están contenidos los sectores ocupados por un fichero. Se puede determinar este número de pista-sector de este sector y leerlo en el buffer, en su caso. Digo que "en su caso", porque un sistema operativo realmente eficiente (como en nuestra gestión relativa) sólo leerá un bloque en el buffer si resulta realmente necesario. Si el mismo bloque hubiese sido leído anteriormente, una nueva lectura resultaría evidentemente supérflua.

Hemos de determinar ahora la posición en la que comienza el juego de datos en nuestro sector calculado. También para ello es esencial la longitud de record. Desde el punto de vista matemático, este cálculo es elemental: se trata de una función normal de resto, también llamada función de MODULO. Calcularemos la posición en el bloque tal y como sigue:

$$7392 - \text{INT}(7392/512)*512 = 224$$

Así que el carácter 224 en el buffer es, en consecuencia, el primer carácter del juego de datos 77. Así es como fue realizada la gestión relativa de datos en el CPC. En cualquier caso, con una pequeña limitación en lo que a la

longitud de record se refiere; pero para ampliar esta información, vea el capítulo 5.3.

1.6.5 ¿Cuándo es lògico el almacenamiento relativo?

Esta es, sin duda, una pregunta interesante, si es que la técnica de almacenamiento relativo le ha hecho "tilin". Hay que admitir que el almacenamiento relativo es muy cómodo. Pero deberia evitar usar el almacenamiento relativo para cualquier pequeño problema.

No deberia olvidar nunca que, por regla general, la programaciòn de ficheros relativos lleva algo más de trabajo de programaciòn. Para la resoluciòn de pequeños problemas tal trabajo de programaciòn resultaria desproporcionado; ello por cierto afecta de igual modo a la velocidad de ejecuciòn.

Así que si desea almacenar pequeñas tablas, o cosas por el estilo, el almacenamiento secuencial es el más indicado para ello. El almacenamiento relativo empieza a ser interesante a partir de unos 50 juegos de datos. El almacenamiento relativo de datos también vale la pena cuando resulte muy difícil, o imposible por completo, retener todos los datos en memoria.

En grandes ficheros personales o de direcciones, el almacenamiento relativo es, con toda seguridad, ineludible. Todo programa de una cierta envergadura de gestión de ficheros como por ejemplo DBASE o bien DATAMAT trabajan exclusivamente con ficheros relativos. Por ejemplo en la ordenaciòn de un fichero siguiendo un cierto criterio, el almacenamiento relativo es, por exigencias de espacio de memoria, la única soluciòn posible.

Antes de afrontar la resolución de un gran problema mediante almacenamiento relativo, deberia "entrenar" un poco. Se presta a ello un pequeño programa que almacena direcciones,

las carga y las corrige. Cuando haya asimilado todo esto, estará en condiciones de verselas con juegos de datos mayores y con mayores cantidades de juegos de datos. Sería lamentable que apareciera un error en momento dado y tuviera que constatar que se sobreescribieron involuntariamente juegos de datos.

1.6.6 Los ficheros ISAM

Se puede afirmar sin temor que los ficheros ISAM son los tipos de fichero que, en la práctica, aparecen con mayor frecuencia. ISAM significa:

Indexed Sequential Access Method

Ha supuesto bien: el "metodo de acceso secuencial indexado" es una mezcla de almacenamiento relativo y almacenamiento secuencial de datos. Para aclarar el concepto "Index", hemos de adentrarnos un paso más en el mundo de la técnica de gestión de ficheros:

Seguro que aún se acuerda de nuestro juego de datos, que contiene cuatro campos de datos: nombre, calle, población y teléfono. Supongamos que tenemos 100 juegos de datos almacenados en este fichero. ¿Qué tendría que hacer, si desease hallar el juego de datos de un cliente, conociendo únicamente su apellido? Exacto: el programa debería - suponiendo que el fichero estuviese ordenado alfabéticamente según el campo de datos apellido - recorrer todos los juegos de datos desde el primero hasta aquél que coincida y comparar el campo de datos apellido con el apellido buscado. Apenas se puede apreciar ventaja alguna respecto a un fichero secuencial.

Existen procedimientos de búsqueda, en los que se optimiza algo el proceso de búsqueda, como por ejemplo en el árbol

binario de búsquedas. Pero siguen llevándose a cabo algunos accesos (innecesarios) a disco. Se pensó después en indicar un campo de datos del juego de datos como "campo clave", en inglés se le llama "key" o simplemente "index". Este campo índice ha de residir (es decir, ha de estar siempre) en memoria. La mejor elección para ello es el campo que se use más frecuentemente como criterio de ordenación o para la búsqueda.

Junto al contenido de este campo de datos se conserva también en memoria, naturalmente, el número de record correspondiente. De este modo se puede acceder directamente en el disco al juego de datos sin tener que leer juegos de datos superfluos.

La ventaja de este procedimiento es clara: por un lado se ahorra memoria, al no conservar en memoria todo el juego de datos. En lugar de $96*100=9600$ bytes necesitamos sólo

$$(35 + 1) * 100 = 3600$$

bytes. Este resultado surge de 35 bytes para el campo clave más un byte para el número de record, multiplicado por los 100 juegos de datos.

La segunda ventaja substancial es el enorme ahorro de tiempo. El ordenador puede comparar una cadena de caracteres mucho más rápidamente que acceder cada vez al disco.

Esta lista de palabras clave es de hecho reordenada cada vez que es completada, borrada o corregida; en caso contrario, un juego de datos podría llegar a no ser hallado. Al finalizar el programa, esta lista de índice es simplemente almacenada como fichero secuencial. El fichero relativo, por tanto, no ha de ser redispuesto, en procesos de ordenación, entre otros.

Evidentemente, pueden crearse más ficheros de índice, de modo que se disponga en cada caso del procedimiento de acceso "óptimo". Pero en la práctica es más que suficiente un único campo de índice. En caso de que se desee buscar por una vez según otro campo de datos, basta con un poco más de tiempo de espera.

1.7 Los distintos ordenadores Amstrad

El CPC 466, el CPC 664 y el CPC 6128 son originarios de la casa Amstrad y son más o menos compatibles entre sí. Los programas BASIC que funcionan en el CPC 466, funcionan también en los otros dos aparatos Amstrad, el CPC 664 y el CPC 6128 pero, lamentablemente, a la inversa no es así.

El CPC 664 y el CPC 6128 tienen un BASIC idéntico, la versión 1.1. Estos dos ordenadores pueden intercambiarse programas BASIC. Pero se le pide mucho cuidado al programador en lenguaje de máquina: los tres aparatos ofrecen aquí un absoluto caos de RAM y ROM. Por suerte, las direcciones de la RAM y de la ROM del floppy no se vieron afectadas por las modificaciones de las distintas versiones, de modo que todo lo relatado en el presente libro resulta válido para los tres aparatos por igual.

Junto con el CPC 466 y el CPC 664 se suministra el CP/M 2.2. Dado que el CPC 6128 es un ordenador de 128 Kbyte, junto con éste se suministran tanto el CP/M 2.2 como el CP/M 3.0. Hallarás una descripción más detallada de ambas versiones de CP/M así como explicaciones de las diferencias y de la utilización en el libro de entrenamiento de CP/M para el CPC.

He aquí una pequeña tabla que ha de dar una idea de las diferencias entre los tres aparatos Amstrad-CPC:

	CPC 464	CPC 664	CPC 6128
Floppy incorporado	N O	S I	S I
Memoria (BASIC)	64 (48)	64 (48)	128 (48)
Versión CP/M	2.2	2.2	2.2 & 3.0
Versión BASIC	1.0	1.1	1.1

1.7.1 CPC 664 y CPC 6128 y errores de disco

Ante la aparición de un error en el CPC 464 - y ésto lo hemos aprendido - el programa en ejecución se interrumpía ineludiblemente, mostrando en pantalla el mensaje de error. Algo muy desagradable y nada cómodo. De esto se dieron cuenta incluso la gente de Locomotive Software, que llevaron a cabo, y a modo de consuelo (apenas es eso), una pequeña modificación en el BASIC.

Ante la aparición de un error en la versión 1.1 de BASIC en el manejo de disco, se puede impedir la interrupción del programa mediante el conocido comando ON-ERROR-GOTO. La variable de error ERR contiene entonces el código de error 32. Sabe entonces, que se trata de un error de disco.

Se le dió al programador BASIC, además, la variable de error DERR, que debe dar información precisa acerca del error de disco. Pero lamentablemente, este número de error proporcionado es claramente insuficiente. Además, el texto de error sigue apareciendo en pantalla. En caso de que se trate de la pregunta

Retry, Ignore or Cancel?

ha de contestar, antes de poder proseguir. Claro - un pequeño paso adelante, pero muy incómodo aún, pues: ¿qué ha de pensar el usuario cuando aparecen en pantalla mensajes con los que no debería toparse?

Así que, y pensando en la compatibilidad: si escribe un programa, que a ser posible no se limite a un disfrute propio, sino que funcione en tantos aparatos como resulte posible, no puede hacer uso en absoluto de la variante ON-ERROR-GOTO ni de la variable de error DERR, ya que le traería complicaciones con los CPC 464. Para este tipo de usos se ofrecen las ampliaciones de BASIC impresas en el capítulo 5.2, que funcionan en los tres aparatos. En un CPC 664 ó en un CPC 6128 se notificaría un 32 ante un error de disco, en el CPC 464 un 18. Con ello se puede ir tirando. Se obtiene además el mensaje de forma clara y no aparece en pantalla.

1.7.2 Otras diferencias entre el CPC 664 y el CPC 6128

Una ampliación racional del BASIC de Amstrad es la siguiente, que no representa de hecho una ampliación de los comandos de disco en pleno sentido, sino simplemente una modificación de los comandos CALL y RSX y sus respectivas listas de parámetros.

Recordará que en el presente libro siempre hemos predicado que, por ejemplo para borrar un fichero, ha de disponer el nombre de fichero en una variable de cadena, para poder transferir luego el puntero a la variable:

```
A$ = "PRUEBA.BAS"  
IERA,A$
```

En el Amstrad CPC 664 y en el CPC 6128 puede introducir el comando también de este modo:

```
IERA,"PRUEBA.BAS"
```

o bien

```
A$ = "PRUEBA.BAS"  
IERA,A$
```

Como queda dicho, una racional ampliación del BASIC de Amstrad, ya que el manejo gana mucho en sencillez y claridad. Pero aquellos entre ustedes que escriben programas que han de funcionar bajo los tres ordenadores, no pueden permitirse este lujo. El CPC 464 se interrumpiría con las dos últimas variantes.

Esta abreviación del comando RSX IDEL es naturalmente válida también para todos los demás comandos de disco RSX:

```
IERA, "nombre de fichero"  
IREN, "nombre de fichero1", "nombre de fichero2"  
IDIR, "expresión de cadena"  
IDRIVE, "unidad"
```

Con lo cual habremos aclarado las diferencias de los tres ordenadores CPC, en lo que a la programación de disco se refiere.

CAPITULO 2:**Programaciòn del floppy para avanzados****2.1 Los vectores del DOS**

Tras aclararle en los capítulos anteriores las posibilidades, tanto del AMSDOS como del CP/M, para el uso 'normal', en los capítulos siguientes aprenderà algo sobre el modo de trabajo del DOS y la utilizaciòn del mismo en lenguaje màquina. Por ello nos basamos en la hipòtesis de que el lector posee unos conocimientos minimos de programaciòn en còdigo màquina del Z80. En caso de que no disponga usted de estos conocimientos, algunos de los conceptos de las pàginas siguientes le resultaràn incomprendibles. Pero no cierre aun el libro. Aunque algunos puntos puedan quedar momentàneamente confusos, resta mucha informaciòn en las pàginas siguientes que podrà serle útil sin mès.

A grandes rasgos, en la programaciòn de la unidad de disco del CPC podemos distinguir tres niveles de programaciòn. El nivel superior es el de mès sencilla comprensiòn. En este nivel, todas las tareas son llevadas a cabo por parte del DOS. De hecho, se trata del equivalente en lenguaje màquina a las órdenes BASIC, que ya conoce del capitulo anterior.

El nivel medio de la programaciòn de la unidad de disco es el nivel que seria inimaginable sin un conocimiento exacto de la ROM del AMSDOS. En este nivel disponemos de rutinas, que van de muy sencillas a realmente complejas, desde la simple conexiòn del motor del floppy hasta la lectura y escritura de datos en disco. Describiremos las rutinas existentes en el DOS, dado que con ellas es posible realizar cosas muy interesantes, imposibles desde BASIC.

El tercer modo posible de programar el floppy consiste casi en programar 'a golpe de calcetin'. Seguro que únicamente pocos especialistas se atreveràn a hacer sus pinitos en este nivel, ya que en él las funciones del controlador de disco han de ser programadas directamente. Pero, dado que dichas funciones se hallan ya de algún modo en el DOS, esta

programación se reducirá a algunos casos especiales. Entraremos también en este tipo de programación, y explicaremos ampliamente las posibilidades del controlador de floppy.

Pero comencemos por el principio. Los sistemas operativos de los tres ordenadores CPC (464, 664 y 6128) no están originalmente dispuestos para el manejo de un floppy. Como único medio externo de almacenamiento conocen el cassette. Sólo a partir de la ROM del AMSDOS (o de otra ROM externa de floppy) resulta posible el funcionamiento de un floppy. En caso de que la ROM no esté disponible en el momento de la conexión del aparato, las órdenes LOAD, SAVE, OPENIN, OPENOUT, CAT y las órdenes de manejo de ficheros se dirigen a la unidad de cintas. Pero en caso de que en el momento de la inicialización del CPC la ROM AMSDOS esté disponible, la cuestión cambia sustancialmente. Casi todas las órdenes que anteriormente se referían al cassette, ahora se entienden referidas a la unidad de disco. La única excepción la constituye la orden SPEED WRITE que, como es sabido, no tiene efecto alguno sobre el floppy.

Para esta flexibilidad resulta esencial el principio de los vectores, utilizado por los creadores del CPC. En lugar de efectuar directamente la llamada a la acción deseada del sistema operativo -léase rutina-, la rutina es dirigida a través de una dirección en la RAM, en la que se halla un salto a la rutina del sistema correspondiente. El que la rutina sea dirigida a través del vector sito en la RAM puede parecer a primera vista superfluo. Pero de hecho, este procedimiento posibilita la compatibilidad de versiones distintas del sistema operativo. Sólo mediante este rodeo resulta posible que programas en lenguaje máquina se ejecuten tanto en el 464 como en el 664 o en el 6128, cuando disponen de sistemas operativos con diferencias, en ocasiones, bastante marcadas. Mientras los vectores se hallen en las mismas posiciones de la RAM y las correspondientes subrutinas sean idénticas en cuanto a su función, puede llevarse a cabo prácticamente cualquier modificación en el sistema operativo. La compatibilidad queda asegurada.

Otro punto esencial es que, tambièn como programador, se tengan todos los cables en la mano. De hecho, si bien las rutinas ROM son inalterables, por hallarse los vectores en la RAM, pueden ser modificados sin problemas, de modo que 'apunten' a rutinas propias. Con ello resulta posible modificar toda funciòn que venga referida por medio de vectores en caso de que sea necesario.

Esto es exactamente lo que ocurre cuando el floppy està apagado. En el CPC hay previstos 22 vectores para el servicio del cassette. 13 de ellos reciben un parche ('patch') para el funcionamiento del Floppy, es decir se modifican las direcciones de las subrutinas a llamar. Mediante estos trece vectores son llamadas desde el BASIC todas las funciones del floppy. Pero tambièn en còdigo màquina pueden utilizarse perfectamente estos vectores. Los nombres corresponden a los del CPC-464-Firmware-Manual.

&BC77	CAS IN OPEN	abre un fichero para lectura entrada
&BC7A	CAS IN CLOSE	cierra ordenadamente un fichero abierto para lectura
&BC7D	CAS IN ABANDON	cierra inmediatamente un fichero abierto para lectura
&BC80	CAS IN CHAR	recoge caràcter del fichero de entrada
&BC83	CAS IN DIRECT	lee completamente en memoria un fichero de entrada
&BC86	CAS RETURN	el ùltimo caràcter es retornado (reescrita)
&BC89	CAS TEST EOF	comprueba ¿fin fichero alcanzado?
&BC8C	CAS OUT OPEN	abre un fichero para escritura (salida)
&BC8F	CAS OUT CLOSE	cierra ordenadamente un fichero abierto para escritura
&BC92	CAS OUT ABANDON	cierra inmediatamente un fichero abierto para escritura
&BC95	CAS OUT CHAR	escribe caràcter en fichero de salida

&BC98 CAS OUT DIRECT escribe completamente una zona de memoria en un fichero
&BC9B CAS CATALOG función equivalente a la orden BASIC CAT

Tras haber leido esta tabla deberia olvidar de inmediato los nombres utilizados. Y es que nos parece bastante ilógico indicar con CAS las rutinas correspondientes en relación con el floppy. En lo sucesivo sustituiremos el simbolo CAS por DISC. Con ello se rectifica el nombre de la función.

2.1.1 DISC IN OPEN &BC77

Antes de leer datos de un fichero, éste ha de ser abierto. La apertura de un fichero se lleva a cabo por la rutina DISC IN OPEN. En ello resulta irrelevante si se trata de un fichero ASCII o de un programa. Esta rutina ha de ser llamada de todos modos en cada caso.

Naturalmente, para leer el fichero se esperan algunos paràmetros. El paràmetro más importante es el nombre de fichero. En todas las rutinas DISC se transfieren paràmetros a los registros. Pero dado que un nombre de fichero consta, como es sabido, de ocho caracteres más tres para la extensiòn, el nombre no puede ser transferido por completo a los registros. En lugar de ello se dota al par de registros HL con la direcciòn en la que se halla en memoria el nombre de fichero. El nombre de fichero puede estar en cualquier parte de la RAM, y tambièn en la ROM, o sea en este caso en los primeros 16 K de la memoria del CPC.

Otro paràmetro, que ha de ser transferido al registro B para usar la rutina, es la longitud del nombre de fichero. Esta es una propiedad muy agradable del DISC IN OPEN, dado que el nombre de fichero no ha de ser llevado por su parte a una longitud fija. Así puede entrar nombres de fichero con, por ejemplo, cuatro caracteres de longitud y una extensiòn de dos caracteres. La rutina alargará por si misma el nombre a la longitud requerida. Pero piense siempre en incluir los caracteres de la extensiòn.

Un tercer paràmetro es esperado por la rutina DISC IN OPEN, en el par de registros DE. Aquí ha de incluir una direcciòn de un buffer de 2048 bytes de longitud. En este buffer se escriben los datos a leer. Aquí tampoco se està sometido a ningún tipo de limitaciòn debido a la posiciòn del buffer en la memoria.

Tras transferir los tres paràmetros a los registros, podemos efectuar la llamada a la subrutina. Efectuemos una llamada de este tipo a tìtulo de ejemplo. Queremos abrir el fichero para la lectura bajo el nombre de 'TEST.DAT'. La secuencia de instrucciones tendría el siguiente aspecto en Assembler:

```
100          LD      HL,FILNAM    ;Direcciòn del nombre
110          LD      B,BUFF-FILNAM ;Longitud del nombre
120          LD      DE,BUFF      ;Direcciòn del buffer
130          CALL   DISC-IN-OPEN
140          RET
150 FILNAM:  DEFM  'test.dat'   ;Nombre del fichero
160 BUFF:    DEFS  &0800       ;Espacio para 2K
```

Pero ¿què es lo que hace exactamente la rutina y cuàles son sus resultados?

En primer lugar se comprueba que el nombre de fichero se ajuste a las conocidas reglas formales. Estas incluyen la prohibiciòn de algunos caracteres. Si se halla uno de estos caracteres, el comando se interrumpe. Por otra parte, el nùmero de caracteres del nombre de fichero no puede ser mayor que 15. Esta cifra incluye nombre de fichero, el punto separador y tres caracteres de extensiòn. Un ejemplo de nombre de fichero de este tipo seria: "0A:FILENAME.DAT"

Simultáneamente a la comprobación del nombre de fichero, todas las minúsculas son convertidas en mayúsculas. Los nombres de fichero que sean menores de lo necesario, son completados mediante espacios.

Tras esta comprobación se revisa en el disco si existe un fichero con ese nombre. En caso de que ello no sea así, p.ej. por haber introducido incorrecta o incompletamente el nombre de fichero, el AMSDOS envía el mensaje 'Filename' not found. Obtendrá el mismo mensaje de error si desea cargar un programa que no se halle en el disco introducido.

Independientemente de si el fichero está o no disponible en el disco, en cada caso obtendrá tras la llamada el retorno & los registros de algunos parámetros importantes.

En primer lugar se puede desprender de la lectura del estado de los flags carry (acarreo) y zero (cero) si el DISC IN OPEN llegó a buen término. Para ello se activa el carry flag y se desactiva el zero flag en caso de que el OPEN fuera fructífero. Si, por contra, están desactivados tanto el carry como el zero, es que intentó abrir un segundo fichero para lectura. Una tercera posibilidad es la de que esté, desactivado el carry flag y activado el de zero. Ello significaría que el fichero no habría sido hallado.

Si el OPEN fue fructífero, mediante el valor del acumulador puede determinarse el tipo de fichero. Así, de tratarse de un fichero de programa BASIC, el acumulador obtendría tras la rutina OPEN el valor 0. Un valor de &16 significa que el fichero abierto es un fichero ASCII. Los restantes valores posibles les serán dados en DISC OUT OPEN en forma de tabla.

El par de registros HL contiene tras la apertura del fichero la dirección del File Header, la cabecera de fichero. El concepto de cabecera de fichero se le habrá presentado en escasas ocasiones, por lo que resulta conveniente comentarlo brevemente. Hallará una descripción detallada del montaje del file header en DISC OUT OPEN. La cabecera contiene una cantidad considerable de información relativa al fichero

correspondiente. Así se halla en ella el nombre del fichero, el tipo de fichero, su longitud y también la dirección en donde fue escrito. Casi todos los ficheros que son almacenados en disco, entre ellos por supuesto programas BASIC y en lenguaje máquina, son almacenados conjuntamente con dicha información, la única excepción la constituyen los ficheros ASCII puros, en los que la cabecera de fichero no es conjuntamente almacenada, sino generada por el AMSDOS.

Más información se puede obtener del par de registros DE. Estos le señalan la dirección inicial del fichero. Por supuesto que esta información sólo tiene sentido cuando el fichero consiste en un programa. Cuando el fichero abierto es un programa, el valor en DE suele ser &0170. En programas en lenguaje máquina DE contiene la dirección a partir de la cual fue escrito el programa.

Como última información, el par de registros BC contiene la longitud del fichero. También aquí hemos de resaltar que esta información no afecta a los ficheros ASCII puros. Los ficheros ASCII pueden ser considerablemente mayores que los 64 K representables mediante un par de registros. De hecho, esta información vuelve a tener pleno sentido sólo para el caso de carga de programas.

2.1.2 DISC IN CLOSE &BC7A

Tras la llamada a esta rutina, un fichero abierto para lectura queda cerrado. Tras ello no se puede volver a leer de este fichero. En la práctica se precisa DISC IN CLOSE para leer datos de un segundo fichero. Se cierra entonces mediante DISC IN CLOSE el primer fichero abierto, antes de abrir para la lectura el fichero nuevo con DISC IN OPEN.

Para quien quiera saberlo, tras la llamada a la rutina se puede determinar a partir del flag de carry si fue abierto un fichero para lectura. Si dicho flag está activado, indica que se abrió un fichero.

Esta rutina no precisa de más paràmetros, dado que no se puede abrir más de un fichero de entrada a la vez.

2.1.3 DISC IN ABANDON &BC7D

Esta rutina realiza casi la misma tarea que DISC IN CLOSE. También tras DISC IN ABANDON queda cerrado un fichero de entrada previamente abierto. En la práctica está concebida para cerrar el fichero en caso de error. También se reclamada por el interpretador BASIC antes de cada LOAD, SAVE y CAT. Ello significa que casi siempre hay ficheros abiertos tras estas órdenes.

2.1.4 DISC IN CHAR &BCB0

De hecho, el AMSDOS conoce dos mètodos distintos para leer datos de un fichero. El primero consiste en la llamada a la rutina DISC IN CHAR, que recoge un caràcter del fichero. Esta rutina tampoco precisa paràmetro alguno, dado que sólo puede haber un fichero de input abierto.

El caràcter leido del fichero es transferido al acumulador. Adicionalmente, se indica mediante los flags de carry y zero si la ejecuciòn de la rutina fue fructífera. Si se recogió un caràcter del fichero, el carry flag estarà activado, y desactivado el zero flag. En caso de que intente leer un caràcter en un fichero no abierto, obtendrà como indicaciòn de error los estados carry flag y zero flag desactivados. En el acumulador se halla, ademàs, el mensaje de error &0e. Este simbolo indica que el fichero de entrada no està abierto, mientras se esperaba que lo estuviera.

También cabe considerar el caso de que llegue usted al final del fichero. Como antes, ambos flags estaràn desactivados, pero del acumulador recibirà usted la respuesta &1a, el indicativo de END OF FILE.

Los posibles mensajes de error le serán presentados al final de este capítulo.

La lectura de datos mediante DISC IN CHAR sólo puede ser llevada a cabo de modo secuencial. Así, si ha leido los 100 primeros caracteres pero desea volver a acceder al décimo, ha de cerrar el fichero y abrirlo de nuevo.

2.1.5 DISC IN DIRECT &BC83

Como ya se mencionó, el AMSDOS ofrece dos posibilidades para la lectura de ficheros. La primera consistente en el uso de la rutina DISC IN CHAR ya la vió usted. DISC IN DIRECT le ofrece la segunda forma de acceso a los datos de un fichero. En contraposición a DISC IN CHAR, con la segunda puede ser leído y almacenado en memoria el fichero completo. Por descontado que la principal aplicación de ésta rutina es la carga de zonas de memoria previamente almacenadas. A través de esta rutina son cargados los programas BASIC y en lenguaje máquina.

Contrariamente al caso de DISC IN CHAR, DISC IN DIRECT ha de estar dotado de un parámetro. Consiste en la dirección de carga del bloque de datos, que ha de transferir usted al par de registro HL.

```
100      LD      HL,BUFFER      ;Dirección inicial datos
110      CALL    DISC-IN-DIRECT
120      RET
130 BUFFER: EQU    $           ;Lectura dirigida a aquí
```

Tras la llamada de DISC IN DIRECT, los flags dan indicaciones, en el modo acostumbrado, acerca del éxito o fracaso de la rutina. Como hasta ahora, carry activado/zero desactivado significa que los datos fueron correctamente leídos. Los mensajes de error posibles corresponden a los de DISC IN CHAR.

El par de registros HL contienen además la dirección de cabecera del fichero llamada dirección ENTRY (de entrada). Hallará el significado de esta dirección en la descripción de DISC OUT OPEN

Junto a ambas posibilidades de acceso a los datos, cabe destacar otra característica. De momento ya tiene uno las herramientas para la utilización de las rutinas. Así que se puede leer un programa mediante DISC IN CHAR de otro modo que en BASIC. Pero si se leyó un carácter del fichero de input mediante DISC IN CHAR, el resto no puede ya ser recogido en la memoria mediante la rutina DIRECT. A la inversa, no resulta posible seguir leyendo con la rutina CHAR un fichero leído antes mediante la rutina DIRECT, menos que se haya alcanzado el final del fichero con DIRECT. Pero tampoco debería leerse con DISC IN DIRECT por segunda vez un fichero; podrían perderse datos en memoria.

2.1.6 DISC RETURN &BC86

La aseveración de que con DISC IN CHAR un fichero sólo puede ser elaborado secuencialmente es fundamentalmente verdadera. Sin embargo, con DISC IN CHAR existe la posibilidad de leer caracteres más de una vez. Con DISC RETURN el último carácter leído es reescrito en el buffer y está disponible de nuevo para la lectura. Una vez leídos los 20 primeros caracteres del fichero, tras la vigésima llamada a DISC RETURN en el siguiente DISC IN CHAR se vuelve a leer el primer carácter. En cualquier caso, esta posibilidad tiene sus limitaciones. Y es que, originalmente, fue concebida para un único carácter.

Un ejemplo aclarará dónde se hallan los límites de esta rutina. Imaginemos un fichero con 4000 caracteres. Si hemos leído 2048 caracteres con DISC IN CHAR, el buffer instaurado en el DISC IN OPEN estará 'vacío', hemos leído todos los caracteres. Si ahora hemos leído el carácter 2049 y llamamos repetidamente a la rutina DISC IN RETURN, deberían volver a cargarse de nuevo los 2048 primeros bytes de datos. Pero

esto no ocurre así. En lugar de ello, el indicador abandona la zona de buffer y se dirige al siguiente carácter a leer, de modo que los caracteres leídos después no tienen nada que ver con el contenido del fichero.

De todo ello se desprende que DISC RETURN sólo puede ser usado con muchas limitaciones. La principal utilidad de esta rutina consiste en la comprobación específica de un carácter concreto. Así se utiliza para comprobar el criterio EOF &1a.

2.1.7 DISC TEST EOF &BC89

Esta rutina le permite determinar si ha alcanzado el final del fichero. Para ello se lee el siguiente carácter mediante DISC IN CHAR y comparado con el valor &1a, 26 decimal. En caso de que el valor del carácter leído sea distinto de 26, el carácter es reescrito en el buffer mediante DISC RETURN y se retorna de la rutina con los flags de carry activado y zero desactivado. En caso de que fuera hallado el valor &1a, ambos flags están desactivados.

Esta rutina puede utilizarse finalmente para la lectura carácter a carácter de un fichero, ya que dicha rutina en sí utiliza la de DISC IN CHAR. La utilizaciòn de esta rutina no està permitida con DISC IN DIRECT y conduce a mensajes de error.

2.1.8 DISC OUT OPEN &BCBC

Todas las rutinas descritas hasta ahora parten de la base de que en el disco hay datos. Las siguientes descripciones de rutinas le mostrarán cómo puede almacenar datos y programas en disco desde lenguaje máquina. Al igual que en la lectura de datos, antes de la escritura, el fichero deseado ha de ser abierto. Ello se consigue, mediante la llamada a la rutina DISC OUT OPEN. Como en la lectura, han de transferirse distintos parámetros a la rutina.

Tenemos en primer lugar el nombre de fichero bajo el que han de poderse volver a hallar los datos en el disco. Como es sabido, la dirección del nombre deseado es transferida al par de registros HL. También es preciso transferir, al registro B y al par DE -como en DISC IN OPEN- la longitud del nombre y la dirección de un buffer de 2048 bytes.

```
100          LD      HL,FILANAME    ;Dirección nombre
110          LD      B,BUFFER-FILENAME ;Longitud nombre
120          LD      DE,BUFFER      ;Buffer de datos
130          CALL   DISC-OUT-OPEN
140          RET
150  FILENAME: DEFM  'test.dat'      ;Nombre de fichero
160  BUFFER:    DEFS  &0800        ;Espacio 2048 car.
```

Seguidamente se comprueba la validez del nombre de fichero y, eventualmente, se completa hasta el número requerido de caracteres. Tras ello se sobreescribe la extensión con tres caracteres de dólar y el nombre es buscado en el disco. De hecho, bajo este nombre se dispone un fichero temporal en el que se inscribe posteriormente la extensión. Luego se dispone una cabecera de fichero, cuya dirección está a disposición del usuario en el par de registros HL tras finalizar la rutina. Pero todo ello en el supuesto de que no aparezcan errores en el DISC OUT OPEN. De nuevo puede comprobarlo mediante el estado del flag de carry. Si al final de esta rutina el carry está activado, todo va bien, el fichero está abierto para escritura. Pero si, por contra, está desactivado, es que ha aparecido algún tipo de error.

Un posible error es el de la entrada erronea del nombre de fichero. Tambièn se notifican como errores los de escritura al comprobar el directorio. En tal caso, el contenido de HL no està determinado.

Pero ocupèmonos del caso en el que el fichero de salida fue correctamente abierto. De hecho ahora obtendremos de nuevo la direcciòn de la cabecera de fichero en el par de registros HL. Observemos detenidamente este fichero.

En principio, la cabecera de fichero no es mès que una zona de memoria de 64 bytes. En estos 64 bytes se almacenan los datos mès importantes sobre el fichero. El significado de los distintos bytes de la cabecera es idèntico en lectura y escritura. La estructura de la cabecera es anàloga a la construcciòn de la misma en la utilizaciòn del cassette.

Los primeros 16 bytes contienen el nombre de fichero. En el caso de las cintas, èste podia ser de hasta 16 caracteres, pero en el servicio de discos la cantidad permitida es de solo 8 caracteres. Si le sumamos la extensiòn, de tres como mèximo, seguimos teniendo apenas 11 caracteres. Adicionalmente se incorpora como primer caràcter el nùmero de usuario. Todo ello suma 12 bytes y los cuatro ùltimos caracteres del nombre de fichero en la cabecera del fichero son siempre 0.

Los bytes 16 y 17 no tienen ningùn significado en el caso del floppy. En la utilizaciòn del cassette contienen el nùmero actual de bloque y un indicativo de si se trata del ùltimo bloque del fichero. Debido a la construcciòn radicalmente distinta, estos datos no son idòneos en el AMSDOS.

El siguiente byte, el 18, que es importante incluso en AMSDOS. Contiene el tipo de fichero del fichero. Està codificado en forma binaria, es decir, cada bit en este byte tiene un significado concreto.

El bit 0 indica si el fichero está protegido. En programas que fueron almacenados mediante SAVE "filename", P el bit 0 en el tipo de fichero está activado.

Los bits 1,2 y 3 determinan el tipo de fichero en sí. Si están desactivados los tres, el fichero es binario, es decir una zona de memoria del CPC. El tipo de los ficheros ASCII se genera activando los bits 1 y 2.

Los bits 4 a 7 suelen estar desactivados e indican, según reza el manual del CPC, el número de versión -pero no nos pregunte lo que eso significa-. Tan sólo los ficheros ASCII tienen el número de versión 1, cuarto bit activado.

Tras el DISC OUT OPEN fructífero, el tipo de fichero está puesto en fichero ASCII. Es tarea suya el poner aquí el valor que precisa.

Los bytes 19 y 20 son de nuevo un residuo del servicio de cassettes. Contienen el número de bytes en el bloque de datos actual, y carecen de sentido en el servicio de discos.

De mayor importancia son los dos bytes siguientes, 21 y 22. Contienen la dirección a partir de la que fueron escritos los datos. Por supuesto que este dato no consta aún en el DISC OUT OPEN. Sólo después de la escritura de un fichero mediante DISC OUT DIRECT (ver abajo) le es proporcionado este campo a la cabecera del fichero. En ficheros ASCII sólo hallará el valor 0, dado que en este caso de datos carece de significado.

El byte 23 de la cabecera será siempre &FF. En el servicio de cassette este valor indica que el bloque de datos leído ahora es el primero del fichero. También carece de sentido en el servicio de discos.

Los bytes 24 y 25 contienen la longitud del fichero. Este valor es transferido al par de registros DE tras el DISC IN OPEN. En el caso de ficheros ASCII este valor es siempre 0 dado que su longitud sòlo està limitada por la capacidad de los discos.

Los dos ùltimos bytes utilizados, 26 y 27, contienen la direcciòn de inicio en programas en lenguaje màquina. Si almacena una zona de memoria mediante el comando

SAVE "memoria.bin",b,1000,2000,1200

el valor 1200 serà puesto en forma hexadecimal en estas dos posiciones de memoria. Pero si trabaja con las correspondientes rutinas en lenguaje màquina, ha de poner este valor usted mismo.

Todos los bytes restantes de la cabecera de fichero son puestos a cero por el AMSDOS tras el DISC OUT OPEN, y no vuelven a ser usados. Quedan entonces a su disposiciòn.

2.1.9 DISC OUT CLOSE &BCBF

Tambièn un fichero de (salida) output ha de ser cerrado. En realidad, la necesidad de cerrar un fichero es aùn mayor en la escritura que en la lectura. Ello radica en el hecho de que cada caràcter pasa en primer lugar por el buffer de 2048 bytes creado en DISC OUT OPEN, y no llega de modo directo al disco. En caso de un desbordamiento de este buffer, o sea, si el nùmero de bytes a almacenar pasa de 2048, dicho buffer es escrito en el 'disco. Así que al cerrar un fichero pueden hallarse en el buffer un màximo de 2047 caracteres. Con la llamada a la rutina CLOSE se escribe igualmente en disco el resto existente de este buffer. Y es ahora cuando el fichero se halla completo en el disco. Así que en el caso màs desfavorable se pierden 2047 bytes si cambia el disco con el fichero de output abierto.

Despuès se cambia el nombre de fichero provisional puesto

por DISC OUT OPEN (con la extensión .\$\$\$) por el nombre de fichero original. Caso de que exista bajo este nombre otro fichero anterior, este último es retitulado, obteniendo la extensión.BAK. Si observa alguna vez en el directorio un fichero con esta extensión (.\$\$\$), lo más probable es que se trate de un fichero no debidamente cerrado.

La rutina CLOSE no precisa de parámetros especiales de transferencia dado que, también en la lectura, puede haber únicamente un fichero abierto al tiempo. Al finalizar esta rutina puede volver a determinar, de la mano de los flags de carry y zero si la ejecución de CLOSE fue regular. En tal caso el carry vuelve a estar activado y el zero flag desactivado.

2.1.10 DISC OUT ABANDON &BC92

Si aparece un grave error en la escritura de un fichero, éste puede ser cerrado mediante esta rutina. Un ejemplo de un error de este tipo lo constituiría quizás el mensaje de que el disco está lleno. En tal caso debería efectuar una llamada a DISC OUT ABANDON, ya que los bloques dispuestos hasta ahora en el disco volverían a ser liberados. En el directorio tampoco aparece el nombre del fichero. En un caso así deberían escribirse por supuesto todos los datos de nuevo en un disco con más espacio disponible.

2.1.11 DISC OUT CHAR &BC95

Igual que en la escritura, el AMSDOS dispone de dos caminos distintos para escribir datos en el disco. El primero mediante DISC OUT CHAR. Con esta rutina se escribe en el OPENOUT-buffer el carácter situado en el acumulador. En caso de desbordamiento, los caracteres escritos en el buffer hasta ese momento son salvados en disco, creando con ello nuevo espacio en el buffer.

Casi cualquier carácter deseado puede ser escrito en un fichero mediante esta rutina. Pero si el fichero es posteriormente leido mediante DISC IN CHAR, ha de ser especialmente precavido en la utilizaciòn del carácter &1a, 26 decimal. En caso contrario podría ser tomado en la lectura como criterio de EOF, aunque no sea el fin del fichero.

Aparte de los caracteres a escribir en el acumulador no ha de transferirse ningún paràmetro a esta rutina. Como en las otras rutinas, tras DISC OUT CHAR el carry està activado si el carácter fue correctamente escrito. Si por el contrario los flags de carry y zero estàn desactivados, es que olvidò abrir como es preceptivo el fichero de salida necesario.

2.1.12 DISC OUT DIRECT &BC98

Esta rutina representa la segunda posibilidad del AMSDOS de almacenar datos en un fichero en disco. En contraposiciòn a la escritura de caracteres individuales, esta rutina se emplea para el almacenamiento de zonas de memoria completas del CPC. Para ello han de transferirse a DISC OUT DIRECT distintos paràmetros.

En primer lugar ha de dotarse al par de registros HL con la direcciòn de inicio de la zona de memoria a escribir. A partir de esta direcciòn seràn escritos los datos en el fichero.

El siguiente valor importante es la longitud deseada del fichero. Para ello cargue el par de registros DE con el nùmero de bytes a almacenar. Se sobreentiende que la longitud màxima de un fichero de este tipo es de 64 K correspondientes a 65536 bytes. No son posibles ficheros mayores. Esta magnitud de fichero ya basta para duplicar la

memoria RAM completa del CPC, lo que no suele tener pleno sentido ya que ésta no podrá volver a ser totalmente cargada.

Si lo desea puede transferir la dirección de entrada (ENTRY) al par de registros BC (p.ej, la dirección de inicio de programas en lenguaje máquina). El contenido de este par de registros es escrito en los bytes 26 y 27 de la cabecera de fichero.

Puede transferir un último parámetro al acumulador; el tipo de fichero. Es igualmente escrito en la cabecera en el byte 18.

Al igual que en la lectura de ficheros, en la escritura es importante no variar el método una vez utilizado. O sea que en un fichero no se puede ir saltando de un método a otro. Si escribió un carácter con DISC OUT CHAR, el intentar utilizar finalmente DISC OUT DIRECT conduce a un mensaje de error.

Lamentablemente, tampoco es posible repetir en un fichero el uso de DISC OUT DIRECT. Aun cuando en tal caso el mensaje del AMSDOS sería un OK, posteriormente en el DISC OUT CLOSE aparecería el mensaje 'File not Open as Expected'. La escritura del fichero sería, además, irregular. Así que tras el primer fin de la rutina ha de ser cerrado el fichero de salida. Cada zona de memoria posterior del CPC ha de ser escrita en disco en un nuevo fichero.

2.1.13 DISC CATALOG &BC98

Considerando sólo esta racional posibilidad, el diskette ya aventaja a la utilización de cassettes. En un breve plazo de tiempo obtiene una visión general de los ficheros en la unidad de disco actual. Además se muestra el número de bloques libres en el disco.

Antes de la llamada a DISC CATALOG ha de dar al par de registros DE la dirección de inicio de una zona de memoria intermedia -buffer- de 2048 bytes. Si bien este buffer no resultaría estrictamente necesario y fue incluido sobre todo por motivos de compatibilidad en la utilización de cassettes, los programadores del AMSDOS han incluido en este buffer un detalle de agradecer. Consiste en que, a la llamada de DISC CATALOG se leen del disco los nombres del fichero y se disponen en el buffer junto con la información sobre el número de bloques ocupados. Es más, los nombres de fichero no se escriben en el buffer en la misma secuencia, sino que se ordenan alfabéticamente. Este es el motivo por el que obtiene un directorio tan ordenadito con la orden CAT. Por contra, con el comando DIR obtiene los ficheros en el mismo orden en el que de hecho se hallan en el disco, es decir, desordenados.

2.1.14 Remendando vectores

Para comenzar este capítulo aclaramos exhaustivamente las ventajas de los vectores situados en la RAM. Pero si se le ocurre la idea de utilizar una ventaja esencial cual es la facilidad de variación de las rutinas correspondientes, debería encaminarse a la tarea con algo de cuidado. Y es que el asunto en sí es un arma de dos filos.

Un vistazo al listado de la ROM del capítulo 3 muestra dónde se halla la dificultad. Los 13 vectores muestran tras la inicialización del AMSDOS el mismo contenido. En todos los casos, los 3 bytes de cada vector son como sigue:

```
RST  &18  
DEFW  &A88B
```

Un sencillo bucle con PEEK confirma la aseveración. Con el RST &18, también llamado RST 3, cabe la posibilidad de acceder a cualquier rutina en cualquier zona de la ROM o de la RAM como subprograma, es decir, casi como CALL. Pero para ello no basta con la dirección de 2 bytes tras la orden de restart, dado que mediante 2 bytes se puede representar tan

sólo una dirección, pero no un valor de opción de ROM adicional. La solución al problema es la siguiente: la dirección tras el RST apunta la dirección de memoria en la que se halla la dirección de 3 bytes definitiva.

En nuestro caso hemos de revisar la dirección &A88B para obtener la dirección y el byte de opción de la auténtica rutina DOS. En la dirección &A88B se hallan los valores &30, &CD y &07. Ello significa en una lectura 'normal' la dirección &CD30 en la ROM de expansión 7.

Si se mira ahora la dirección indicada en la ROM del AMSDOS, se hallará una rutina algo peculiar que proporciona la dirección del vector reclamado mediante la manipulación del stack (de hecho, en el stack se halla la dirección del vector +3). A esta dirección se le suma un valor de &10D2 y el resultado es puesto en la parte superior del stack. Finalmente un RETURN conduce a la rutina requerida en sí. ¿Qué pasa?

El motivo de tomar este camino indirecto para el salto de las distintas rutinas está claro si se considera que el RST 3 funciona como un CALL. Pero en un CALL se sitúa en la pila una dirección de salto de retorno, que apunta detrás del vector llamado, en la siguiente anotación en el bloque de saltos. Pero nuestro programa ha de ser llevado al punto que sigue a la llamada al vector. Por ello resulta imprescindible que la dirección de salto de retorno del RST desaparezca del stack.

Precisamente de esta tarea se encarga la rutina en &CD30. Pero la dirección de salto de retorno a olvidar no es simplemente eliminada, sino que se suma al ya mencionado valor &10D2 dando la auténtica dirección de la rutina a llamar. Con ello quedan aclaradas las dificultades al remendar -del inglés patch- las rutinas. Si un RST llega a la dirección &CD30 desde otra dirección como una anotación de bloque de saltos, el resultado de la suma es un valor casi irracional y el bloqueo del sistema o un reset son las consecuencias más probables.

Pero basta de explicaciones; veamos cmo realizar, a pesar de todo, estos patchs de rutinas.

El ejemplo elegido es bastante sencillo. El nico objetivo del patch es mostrar cmo trabaja uno de estos patchs. Para este ejemplo desviaremos el vector CAT.

```
100 INIT:    LD      HL,&BC9C ;direccin del vector CAT
110          LD      TEMP,HL ;almacenar
120          LD      HL,PATCH ;nueva direccin del RST
130          LD      &BC9C,HL ;realizar patch
140          RET     ;todo bien
150 PATCH:   EQU    $      ;posible lugar para su rutina
160          LD      HL,TEMP ;direccin original
170          LD      &BC9C,HL ;volver a anotar
180          CALL   &BC9B ;vect. CAT con direc. correcta
190 PATCH1:  EQU    $      ;otro lugar para su rutina
200          CALL   INIT   ;refijar para prxima llamada
210          RET     ;ya se acab
```

Como ve, de hecho no hacemos nada aparte de desviar el vector. En cualquier caso, de este modo tiene usted la posibilidad de acceder, tanto antes como despues de la rutina.

Un ejemplo prctico del mtodo utilizado es la reparacin de MERGE y CHAIN MERGE. Quiz se decida a desensamblar estos cortos programas para averiguar cmo est programado el acceso a DISC IN CHAR.

2.2 Las ampliaciones de rdenes del AMSDOS

Mediante las rutinas descritas en el captulo anterior no se han agotado, ni de lejos, las posibilidades del programador. Tambin las rdenes de la ampliacin, indicada en BASIC mediante la barra vertical I 'SHIFT @', pueden ser utilizadas sin dificultad en lenguaje mquina. Ahora nos ocuparemos concretamente de cmo ocurre esto.

Ya conoce los comandos posibles de capítulos anteriores y también de su manual del floppy. He aquí de nuevo una visión de los comandos disponibles:

CPM	A
DISC	B
DISC.IN	DRIVE
DISC.OUT	USER
TAPE	DIR
TAPE.IN	ERA
TAPE.OUT	REN

Al contrario de lo que ocurría en las rutinas DISK, en los comandos de las ampliaciones de órdenes no hay vectores, de modo que no se puede saltar directamente a ellas. Para las ampliaciones de órdenes se requieren medidas especiales para 'poner en marcha' la acción deseada. Así que confiemos en el mecanismo.

2.2.1 Programación en ensamblador de las ampliaciones

Si los comandos de cierta ampliación de órdenes han de ser ejecutados, en primer lugar ha de conocerse la dirección de la rutina. Dado que estas rutinas pueden hallarse tanto en la RAM en forma de ampliación RSX como en ROMs en los 16K superiores del CPC, la eventual dirección de ROM ha de ser proporcionada. Esto lo realiza una rutina especial del Kernel del CPC, cuando le damos el nombre de la ampliación de comandos deseada. Para ello ha de ponerse la dirección del nombre en el par de registros HL. Ha de dar este nombre en mayúsculas. El último carácter del nombre se indica activando el séptimo bit de dicho carácter. Exactamente del mismo modo están registrados en ROM y RAM todos los nombres de las ampliaciones de órdenes.

Si tenemos el nombre en RAM en la forma deseada y HL está dotado por la dirección de dicho nombre, basta la llamada a la rutina KL FIND COMMAND.

KL FIND COMMAND es una rutina del kernal del CPC y tiene un vector en RAM en la dirección BCD4. Tras ejecutar esta rutina, nos son devueltos los siguientes paràmetros:

Si el carry flag està desactivado, es que no fue hallado el comando correspondiente. O bien no entrò correctamente el nombre de la ampliacion de órdenes, o bien la ampliacion no ha sido aún inicializada.

La ùltima situaciòn puede aparecer con el floppy si enciende en primer lugar el CPC y despues el floppy. Pero pulsando simultaneamente las tres teclas CTRL, SHIFT y ESC se lleva a cabo un reset en el que se 'le pega un timbrazo' a la ampliacion de órdenes, que està a nuestra disposiciòn a partir de entonces.

Si, por el contrario, el carry flag se halla activado tras la rutina KL FIND COMMAND, obtendrà la direcciòn de la rutina deseada en el par de registros HL. La direcciòn de ROM precisa se halla en el registro C.

El proporcionarla precisamente a este registro fue una elecciòn meditada de los programadores del sistema operativo del CPC, dado que existe una rutina del kernal llamada KL FAR PCHL, que puede llamar a cualquier direcciòn de cualquier ROM posible o de la RAM como subprograma. La direcciòn de la rutina ha de ir para ello a HL, la direcciòn ROM precisa en el registro C. Justamente ahí se hallan tras KL FIND COMMAND los valores que necesitamos, de modo que ya nada se interpone a la llamada.

Observemos una vez este mecanismo con ayuda del comando IDIR, que, como es sabido, no precisa de màs paràmetros, antes de meternos con comandos que requieren paràmetros en forma de variables numèricas o de cadena.

```
100      LD    HL,COMMAND      ;Direcciòn del comando
110      CALL  KL-FIND-COMMAND ;Direcciòn &BCD4
120      RET   NC              ;Comando no hallado
```

```

130          XOR   A           ;Ningún paràmetro
140          CALL  KL-FAR-PCHL ;Direccióñ &001B
150          RET
160 COMMAND: DEFM  'DI','R'+&80 ;Nombre del comando

```

Las dos rutinas del kernel utilizadas son realmente fantásticas en sus prestaciones. Ya nadie tendrá que sufrir con largas tablas de direcciones de todas las rutinas precisas. Es totalmente suficiente conocer el nombre de la rutina.

La programaciòn serà un poco mès laboriosa si se han de transferir paràmetros a la ampliaciòn de órdenes. Pongamos por caso que hemos de transferir valores numèricos enteros. Este es el caso de la orden IUSER. Son vlidos los nmeros USER de 0 a 15. Veamos un programa ejemplo para la llamada del comando USER, antes de comentar las particularidades de la transferencia de paràmetros:

```

100          LD    HL,COMMAND ;Direccióñ del comando
110          CALL  KL-FIND-COMMAND ;Direccióñ &BCD4
120          RET   NC           ;Comando no hallado
130          LD    A,1           ;Transferir un paràmetro
140          LD    IX,NUMBER    ;Nmero de User buscado
150          CALL  KL-FAR-PCHL ;Direccióñ es &001B
160          RET
170 COMMAND: DEFM  'USE','R'+&80 ;Nombre del comando
180 NUMBER:  DEFW  0004

```

En primer lugar ve usted que la rutina IUSER proporciona al acumulador el nmero de paràmetros transferidos. Cualquier valor distinto de uno lleva a la salida de 'Bad command'.

El registro IX muestra el nmero de usuario deseado, el cuatro en nuestro ejemplo. Dado que slo pueden ser transferidas palabras de 16 bits, en la linea 180 el valor

de NUMBER ha de ser definido con la instrucción DEFW como valor de dos bytes (16 bits).

A la vista està que la programaciòn no ha mejorado ostensiblemente. Algo mès de trabajo conlleva el transferir cadenas a la ampliaciòn de órdenes. Tal es el caso en las órdenes IERA, IREN y IDRIVE.

Como ya sabe, las cadenas no pueden ser transferidas directamente a órdenes de la ampliaciòn de órdenes. En lugar de ello se transfiere el puntero de variable, que usted obtiene con la funciòn 'arroba=0'. El puntero de variable es en si un indicador del llamado descriptor de cadena. La arquitectura del descriptor de cadena y la gestiòn de cadenas en BASIC ya deberia resultarle conocida. Por motivos de espacio no podemos extendernos en mayores explicaciones.

Consideremos primero el caso sencillo, con una sola cadena por paràmetro, el comando IERA. En nuestro ejemplo queremos borrar un fichero de nombre 'DIRECCIO.DAT'.

```
100      LD      HL,COMMAND      ;Direcciòn del comando
110      CALL    KL-FIND-COMMAND ;Direcciòn &BCD4
120      RET     NC              ;Comando no hallado
130      LD      A,1              ;Transferir un paràmetro
140      LD      IX,VARPTR        ;Descriptor de variable
150      CALL    KL-FAR-PCHL      ;Direcciòn es &001B
160      RET
170 COMMAND: DEFM   'ER','A'+&80  ;Nombre del comando
180 VARPTR:  DEFW   DESCRIPT  ;Direcciòn descriptor
190 DESCRIPT: DEFB   12          ;Longitud de la variable
200 FILNAM:  DEFW   FILNAM    ;Direcciòn del nombre
210 FILNAM:  DEFM   'DIRECCIO.DAT' ;Fichero a borrar
```

Algo mès útil todavia resulta esta posibilidad si se precisan dos cadenas, como en el caso del comando IREN. El siguiente ejemplo le muestra còmo puede cambiar el nombre del fichero 'DIRECCIO.DAT' por el de 'DIRECCIO.ALT'.

```
100      LD      HL,COMMAND      ;Dirección del comando
110      CALL    KL-FIND-COMMAND ;Dirección &BCD4
120      RET     NC             ;Comando no hallado
130      LD      A,2            ;Trans. 2 n.fichero a
REN
140      LD      IX,VARPTR      ;Descriptor de variable
150      CALL    KL-FAR-PCHL   ;Dirección es &001B
160      RET
170 COMMAND: DEF M  'RE','N' +&80 ;Nombre del comando
180 VARPTR:  DEF W  DESCOLD    ;Dir. descriptor n.anterior
190      DEF W  DESCNEW      ;Dir. descriptor n.nuevo
200 DESCOLD: DEF B  12         ;Longitud de la variable
210      DEF W  OLDNAME     ;Dirección anterior nombre
220 OLDNAME: DEF M  'DIRECCIO.DAT' ;Antiguo nombre fichero
230 DESCNEW:  DEF B  12         ;Longitud de la variable
240      DEF W  NEWNAME     ;Dirección nuevo nombre
250 NEWNAME: DEF M  'DIRECCIO.ALT' ;Nuevo nombre fichero
```

Aunque nosotros hemos dispuesto en el ejemplo todos los datos uno al lado del otro, en la práctica pueden hallarse en cualquier parte de la memoria del CPC. Tampoco el orden tiene importancia alguna. Lo importante es que los punteros estén sobre los descriptores en el orden adecuado e, ineludiblemente, uno detrás del otro en la memoria.

Con los conocimientos de este capítulo ya debería resultarle posible utilizar con lenguaje máquina todas las posibilidades alcanzables desde el BASIC. Esto es, desde luego, algo a considerar, pero es sólo una parte de lo posible. La meta de la programación en lenguaje máquina es precisamente el ampliar las fronteras de las posibilidades existentes en BASIC. Pero aún no hemos llegado a ello.

En qué consisten las posibilidades que se le ofrecen al programador, es lo que averiguaremos en el siguiente capítulo.

2.2.2 Las órdenes 'ocultas' de la ampliación

Al elaborar el listado de la ROM no hallamos en el manual ni un silaba sobre las mencionadas posibilidades. Junto a las 14 conocidas órdenes de la ampliación de órdenes, existen otras 9 órdenes, en parte muy interesantes, que pueden ser activadas del mismo modo que las órdenes hasta ahora descritas. En cualquier caso, estas órdenes no pueden ser activadas desde BASIC como se hacia con, por ejemplo, el comando IDIR.

Todos estos comandos tienen un nombre de una longitud de un carácter. Dichos nombres son &01, &02 ... &09. Dado que en todos los nombres de la ampliación de órdenes ha de ser activado el séptimo bit del último carácter, los nombres 'de facto' son &81, &82 ... &89. Estos nombres no se pueden introducir en programas BASIC. Por ello, la utilización de estas órdenes queda reservada al programador de lenguaje máquina. En algunas órdenes la utilización en BASIC no tendría demasiado sentido, como podrá deducir de la descripción de las distintas órdenes.

2.2.2.1 La orden &81 Message ON/OFF

Esta orden permite desconectar mensajes de errores que pueden aparecer relacionados con las órdenes &82 hasta &89, que aún hemos de ver. Son, en particular, los mensajes de error del controlador de disco que reclaman del usuario la entrada de C, I o bien R por Chancel, Ignore y Retry. Para conseguir esta desconexión de los mensajes de error, el programador ha de transferir al acumulador un valor distinto de cero antes de la llamada.

```

100      LD      HL,COMMAND      ;Dirección del comando
110      CALL    KL-FIND-COMMAND ;Vector &BCD4
120      RET     NC              ;Comando no hallado
130      LD      A,&FF            ;Para desconexión de mensajes
140      CALL    KL-FAR-PCHL    ;Dirección es &001B
150      RET
160 COMMAND: DEFB  &81          ;Nombre del comando

```

Toda la acción de este programa consiste en transferir el valor en el acumulador a la posición de memoria &BE78. Esto sería más rápido escribiendo explícitamente dicha posición de memoria. Pero si por algún motivo, en versiones futuras del AMSDOS fuese variada la dirección de esta posición de memoria, la rutina anteriormente mostrada sería con toda seguridad ajustada en el AMSDOS de modo que se proporcionase la dirección de RAM modificada. Así que, por motivos de compatibilidad con estas posibles versiones posteriores, debería utilizar la rutina anterior cuando no programe para sí mismo.

Esta rutina no es incorporable en el BASIC 1.0 ya que depende usted de mensajes de error en pantalla. El hecho es que el número de error transferido al acumulador no está disponible en el BASIC 1.0. En lugar de ello, el programa sería interrumpido y en pantalla aparecería el lacónico mensaje 'BREAK'. En el BASIC 1.1 del CPC 664 y CPC 6128, sin embargo, puede incorporar esta rutina y preguntar por errores mediante ON ERROR GOTO y la variable del sistema reservada DERR. Pero para programas BASIC resulta más indicado para la intercepción de mensajes de error el programa que encontrará más adelante

2.2.2.2 La orden &82 Drive Parameter

Esta orden permite la modificación de los datos de la unidad de disco. En ellos se incluyen el tiempo de espera hasta que el rotor alcance la velocidad de rotación debida tras encender el motor del disco, también el periodo de tiempo que se ha de esperar tras un cambio de pista, así como el

tiempo remanente de rotación. También se determinan de nuevo en esta rutina los períodos para el HEAD LOAD TIME y el HEAD UNLOAD TIME.

Contrariamente a todas las demás rutinas de la ampliación de órdenes, anteriormente vistas, &82 no puede ser llamada mediante las conocidas rutinas del kernel. Y es que &82 espera hallar en el par de registros HL el comienzo de la tabla para los datos de la unidad de disco. Esto cancela el salto mediante KL FAR PCHL, dado que esta rutina del kernel precisa en HL la dirección de la rutina a la que hay que saltar. Pero hay otros caminos para efectuar la llamada a una orden de una ampliación.

```
100      LD      HL,COMMAND      ;Dirección del comando
110      CALL    KL-FIND-COMMAND ;Vector &BCD4
120      RET     NC             ;Comando no hallado
130      LD      (FARADR),HL      ;Anotar dirección rutina
140      LD      A,C             ;Selección de ROM al acu.
150      LD      (FARADR+2),A      ;Y almacenar también
160      LD      HL,NEWTAB        ;Tabla parámetros unidad
170      RST    &18              ;Va como CALL a rut. des.
180      DEFW   FARADR          ;Vector a dir. FAR (3 B.)
190      RET
200 COMMAND: DEFB   &82          ;Nombre del comando
210 FARADR:  DEFS   3             ;Espacio para dirección
                                  ;de 3 bytes
220 NEWTAB:  DEFS   7             ;Es preciso transferir
                                  ;7 bytes a &82
230 HDUNLD:  DEFS   1             ;Aqui ha de ir el HEAD
                                  ;UNLOAD-TIME deseado
240 HEADLD:  DEFS   1             ;HEAD LOAD-TIME tras
                                  ;página del FDC
```

Hemos introducido de un modo algo distinto al utilizado en las anteriores llamadas al AMSDOS una orden RESTART. El RST &18 supone una versión distinta de la orden CALL. A través de este RST puede efectuarse, similarmente a la rutina KL-FAR-PCHL, cualquier dirección en cualquier ROM o RAM posible. Pero la ventaja del RST &18 consiste en que no se

precisa registro alguno. Mediante esta rutina podemos transferir parámetros incluso al par de registros HL y al registro C para los subprogramas a activar, lo que no era posible con las rutinas del KERNAL utilizadas hasta ahora. Para ello los tres bytes han de ser transferidos a la dirección FAR en la memoria del CPC. Estos tres bytes, es decir, la dirección de dos bytes y el byte de ROM-select han de ser dados en cualquier caso en el orden indicado y conjuntamente.

Veamos ahora el aspecto que ha de tener la tabla requerida.

El primer apunte en la tabla es un valor de 16 bits que define el tiempo de espera necesaria tras la conexión del motor tractor de la unidad. Resulta obvio que el disco no alcanza la velocidad nominal de rotación inmediatamente tras la conexión del motor tractor de la unidad. Ello llega tras aproximadamente un segundo. Correspondientemente, el valor estándar es de 50 decimal, que va siendo decrementado con un ticker-event. Dado que el ticker se verifica cada 1/50 segundos, el tiempo de espera es de un segundo justo.

El segundo apunte en la tabla describe el tiempo de rotación remanente, rotación que sigue al último acceso al disco. También aquí se trabaja con un ticker. Tras la inicialización del AMSDOS, el valor de 16 bits es de 250 decimal. Esto corresponde a un periodo de 2,5 segundos.

En el caso del tercer apunte se trata de un valor de un byte. En el modo estándar, el contenido es un valor de &af. Este valor sólo se precisa para la rutina &86, formateado de una pista, y no debería ser alterado.

El siguiente valor de 16 bits también define tiempos de espera. El byte alto (high) de este valor define el tiempo que ha de aguardarse tras un cambio de pista. El estándar incorporado es de un lapso de 12 ms. Este lapso depende de la unidad empleada.

En la tabla han de ser definidos otros dos tiempos de espera. Son los valores para los HEAD LOAD TIME y HEAD

UNLOAD TIME, que precisa el IC del controlador. Los valores dados en la ROM son de 32 ms para el HEAD UNLOAD TIME y de 16 ms para el HEAD LOAD TIME. Puede hallar el significado preciso de estos valores en la descripción del FDC 765.

La orden &82 es ejecutada de modo automàtico por el AMSDOS tras cada conexiòn o Reset. Tambièn bajo CP/M se efectúa una vez esta funciòn al poner en marcha el CP/M. Bajo CP/M cabe la posibilidad de definir a voluntad los distintos paràmetros a voluntad en el programa SETUP.COM. Hallarà la tabla utilizada tras el Reset en la direcciòn &C5D4.en la ROM del AMSDOS.

2.2.2.3 La orden &83 Disk Format Parameter

Como es sabido, el CPC puede trabajar, de origen, con tres formatos de disco distintos. La orden &83 possibilita determinar el formato del disco introducido en la unidad activa. Para cada formato se halla una tabla en la ROM del AMSDOS. Segùn el valor transferido al acumulador, la tabla precisa es dispuesta en la RAM del CPC. Gracias a ello puede el usuario disponer la tabla correcta en programas en lenguaje mèquina mediante acceso directo. La utilizaciòn en BASIC es posible, pero innecesaria ya que el AMSDOS determina por si mismo estos valores. Incluso si expresa un determinado formato, previo a cada acceso, se proporciona adicionalmente de nuevo el formato.

```
100      LD    HL,COMMAND      ;Direcciòn del comando
110      CALL  KL-FIND-COMMAND;Vector &BCD4
120      RET   NC              ;Comando no hallado
130      LD    (FARADR),HL      ;Anotar direcciòn rutina
140      LD    A,C              ;Selecciòn de ROM al acu.
150      LD    (FARADR+2),A      ;Y almacenar igualmente
160      LD    A,FORMAT         ;Indicativo del formato
                                ;deseado (ver texto)
```

```

170      RST    &18          ;Como CALL a rut. deseada
180      DEFW  FARADR       ;Vector a dir. FAR (3 B.)
190      RET
200 COMMAND: DEFB  &B3        ;Nombre del comando
210 FARADR: DEFS  3          ;Esp. para dir. 3 bytes

```

El programa no se diferencia en modo alguno de los mostrados hasta ahora. Unicamente queda por aclarar què aspecto tienen los indicativos de los diferentes formatos de disco.

La diferencia radica en los nùmeros de sector. En el formato CP/M del AMSDOS, en cada pista hay 9 sectores con 512 bytes cada uno. Estos nueve sectores llevan los nùmeros de sector &41 hasta &49. Esto significa que en cada nùmero de sector, el sexto bit està activado. Por el contrario, en el formato de datos del AMSDOS, los nùmeros de sector van de &C1 hasta &C9. En este caso estàn fijados los bits sexto y sèptimo en cada nùmero de sector. En el formato CP/M de IBM los nùmeros de sector van de &01 hasta &08. Es sabido que en este formato sólo hay ocho sectores formateados por pista. Con ello tenemos un criterio para distinguir los tres formatos. Transfiera el valor 0 a la rutina &83 (o cualquier otro valor de bits 6 y 7 nulos), y la tabla de paràmetros para el formato CP/M de IBM serà dispuesta en la RAM. El formato CP/M del AMSDOS se consigue transfiriendo bit 6 activado y bit 7 desactivado. Para ello sirve tanto &40 como &73. Si transfiere el valor con sus dos bits superiores activados (p.ej. &C0 o bien &FF), conseguirà el formato de datos del AMSDOS.

2.2.2.4 La orden &84 Read Sector

No hay que infravalorar la capacidad de esta orden. Permite leer directamente cualquier sector del disco.

Con ello deja usted de estar sujeto a las conocidas estructuras de datos, pudiendo construir a voluntad estructuras propias como, por ejemplo, ficheros relativos o ficheros ISAM, dado que la orden siguiente, &85, permite

escribir directamente en cualquier sector del disco. Mediante estas dos órdenes, todos los bytes del disco estàn a su entera disposiciòn.

Para poder leer un sector determinado mediante &84, han de darse algunas indicaciones a la rutina.

En primer lugar, la unidad de disco deseada. Esta informaciòn ha de ser depositada en el registro E. Un valor de 0 en el registro E selecciona la unidad A, un 1 selecciona la unidad B.

Por supuesto que, ademàs, hemos de notificar a la rutina el sector deseado. El registro elegido para este paràmetro es el registro C. Ha de indicarse el nùmero de sector efectivamente existente en el disco. Asi que si desea leer el primer sector de una pista escrita en formato de CP/M del AMSDOS, el nùmero de sector correcto es &41.

Se precisa, ademàs, la pista deseada del disco. El nùmero de pista es dado al registro D. En el AMSDOS existen los nùmeros de pista de &00 a &27 (39 decimal).

Con ello hemos transferido todos los paràmetros importantes a los registros adecuados. ¡Pero vayamos poco a poco! Hemos de decidir en què posiciòn de memoria han de disponerse en la lectura los 512 bytes del sector. Este dato ha de ser transferido al par de registros HL. Con ello tenemos todos los paràmetros en orden, y los datos pueden ir llegando.

```
100      LD    HL,COMMAND      ;Direcciòn del comando
110      CALL  KL-FIND-COMMAND ;Vector &BCD4
120      RET   NC              ;Comando no hallado
130      LD    (FARADR),HL      ;Anotar direcciòn rutina
140      LD    A,C              ;Selecciòn de ROM al acu.
150      LD    (FARADR+2),A      ;Y almacenar igualmente
160      LD    E,DRIVE          ;0/1 para disco A/B
170      LD    D,TRACK           ;0 - 39 por pista deseada
180      LD    C,SECTOR          ;No.sec.con format-offset
190      LD    HL,BUFFER          ;512 bytes datos sector
200      RST   &18              ;Va como CALL a rut.des.
```

```
210      DEFW  FARADR      ;Vector a dir. FAR (3 B.)
220      RET
230 COMMAND: DEFB  &84      ;Nombre del comando
240 FARADR: DEFS  3        ;Esp. p. direcc. 3 bytes
```

Por el momento no pretendemos escribir un monitor completo de disco, pero el siguiente programa BASIC dispone de la función 'lectura de sectores'. Tecleelo con calma, ampliélo y analicelos, para ganar confianza en el manejo de las órdenes. Este breve programa supone el armazón del ya mencionado monitor de disco.

Antes de haberse convencido totalmente del correcto funcionamiento del programa, deberia hacer uso de la protección de escritura de su disco. Basta un byte incorrecto, el de comando, para que el sector elegido no sea leido, sino sobreescrito con el contenido del buffer. Si sobreescribe de este modo el primer sector del directorio de su disco master de CP/M con ceros, espero que previamente haya hecho una copia del mismo. En caso contrario, se perdieron con toda seguridad 16 programas del disco. Por su propio interés, grábese en la mente este consejo:

!!! BACKUP, BACKUP, BACKUP !!!

```
100 DEFINT a-z
110 MEMORY &A000 -1
120 FOR adress= &A000 TO &A01C
130 READ byte
140 POKE adress,byte
150 check = check + byte
160 NEXT adress
170 IF check <> 2941 THEN PRINT"Error en DATA$ !":END
180 MODE 2
190 INPUT "Output stream (0/8)";dev
200 INPUT "Unidad (0/1)";drive
210 INPUT "Pista (0-39)";track
```

```
220 INPUT "Sector          (1-9)";sector
230 POKE &A020,drive
240 POKE &A021,track
250 POKE &A022,sector+64
260 CALL &A000
270 MODE 2
280 lincnt = 0
290 FOR i = &A030 TO &A030+511
300 IF lincnt = 0 THEN PRINT #dev," ";HEX$(i-&A030,4);"
310 PRINT #dev,HEX$(PEEK(i),2);";:lincnt=lincnt+1
320 IF lincnt = 16 THEN lincnt=0
330 a=PEEK(i):a=a AND 127
340 IF a<32 OR a=127 THEN t$=". " ELSE t$=CHR$(a)
350 lin$=lin$+t$
360 IF lincnt = 0 THEN PRINT #dev,lin$:lin$=""
370 NEXT
380 PRINT" <SPACE> para seguir "
390 IF INKEY (47) THEN 390
400 GOTO 180
410 DATA &21,&1c,&a0,&cd,&d4,&bc,&22,&1d
420 DATA &a0,&79,&32,&1f,&a0,&21,&20,&a0
430 DATA &5e,&23,&56,&23,&4e,&21,&30,&a0
440 DATA &df,&1d,&a0,&c0,&b4
```

El manejo de este programa se entiende por si mismo. La transferencia de las indicaciones sigue el principio del buzón, es decir, escribimos los valores en posiciones de memoria concretas, de donde son recogidos por el programa en lenguaje máquina. Ello es necesario, dado que lamentablemente no existe modo alguno de cargar desde el BASIC los contenidos de los registros del Z80 directamente con algún valor.

Tras haber asignado mediante POKE los paràmetros precisos a sus posiciones de memoria, se salta a la rutina en lenguaje máquina mediante CALL. El sector indicado es buscado en el disco y los datos se escriben en un buffer de 512 bytes, a partir de &A030. De aquí lee el programa BASIC los bytes,

que son llevados a pantalla en forma hexadecimal. Si lo desea puede desviar la salida a una impresora conectada.

2.2.2.5 La orden &85 Write Sector

La lectura de datos ya es, en si, bastante interesante. La rutina anterior permite echar un interesante vistazo, p.ej. al directorio o a las pistas del sistema 0 y 1 de un disco de CP/M. Pero podemos más aún. La escritura de datos en cualquier sector del disco que deseemos es, con la orden &85, igual de sencillo que la lectura con &84. Incluso los paràmetros necesarios, nùmero de unidad, pista, sector y direcciòn de buffer de datos son transferidos a los mismos registros que en la lectura. Con ello se reduce la variaciòn del anterior programa impreso a algunos bytes. Solamente han de ser variados el nùmero de la orden, el ùltimo elemento DATA en el programa BASIC y, por supuesto, la suma de comprobaciòn.

En plan experimental, puede tomar un disco recien formateado, e intentar escribir un sector. Para ello introduzca, mediante POKE, algunos bytes en el buffer de sector y escriba éste finalmente. Mediante la rutina de lectura impresa puede comprobar si todo ha ido bien.

2.2.2.6 La orden &86 Format Track

Esta orden es, en cierto modo, para especialistas. Permite el formateado de una ùnica pista en el disco. Con ayuda de la orden &86 puede incorporar rutinas de formateado en sus programas, de modo que el usuario no se vea necesitado del programa en CP/M 'FORMAT.COM'.

Antes de que expliquemos esta orden 'rescatada' de la ampliaciòn, hemos de aclarar el modo en el que el FDC 765 formatea una pista. Para no adelantarnos a la posterior

descripciòn del FDC 765, desvelemos por ahora lo siguiente:

El floppy-controller es, en relaciòn al formateado de discos, extraordinariamente 'agradable al uso'. Le bastan algunos bytes, pocos, y se encarga del resto del 'trabajo sucio', como la generaciòn de sumas de comprobaciòn, el realizar las distintas marcas ID y los llamados GAPs. Ademàs vigila cuàndo es marcado el comienzo de pista por la perforaciòn de indice y reconoce si el disco està protegido frente a escritura.

Si este minàsculo elemento ha de formatear un track, es decir, una pista, precisa en primer lugar el comando correspondiente. Este comando le dice (entre otras cosas), en què unidad (drive) y què pista ha de formatear, la longitud de los sectores a generar y què valor ha de emplear como fillerbyte. El fillerbyte, o byte de relleno, es el valor que se halla en todos los sectores como dato tras un formateado correcto.

Parece evidente que para formatear, el motor de la unidad ha de estar en funcionamiento. Tan pronto como se reconoce la perforaciòn de indice tras el comando, comienza el autèntico proceso de formateado. El FDC espera ahora del procesador cuatro bytes por cada sector a formatear. Para cada sector han de indicarse el nùmero de pista, de cabezal, de sector y la magnitud del sector. Debido a que para cada sector se ha de dar tambièn el nùmero de sector, pueden llevase los sectores al disco en una secuencia determinada por el programador. Con ello puede acelerarse considerablemente el posterior acceso.

Pero ya seguiremos luego con esto. Veamos què tiene que ver todo esto con el comando &86.

El comando &86 espera paràmetros en distintos registros, de modo parecido a las órdenes de escritura y lectura de sectores anteriormente descritas. En primer lugar, estaría la pista deseada. Este valor, como es sabido, es transferido al registro D. En el registro E del Z80 ha de hallarse el

número para la unidad de disco deseada. El registro C recibe el número del primer sector a formatear y el par de registros HL es utilizado como puntero de una tabla.

Los tres primeros registros y sus funciones no se diferencian de aquellos de las rutinas anteriormente descritas. Pero también el par de registros HL tiene una función comparable a la de la escritura de datos. En la tabla direccionada mediante HL se hallan cuatro bytes por cada sector a formatear, los ya mencionados valores para pista, cabezal y unidad, número de sector y magnitud del sector. El siguiente programa ejemplo le muestra cómo puede formatearse una única pista.

```
100          ORG    &5000      ;FORMATEAR PISTA COMPLETA
110          LD      E,DRIVE   ;Dir. inicial rutina formateado
120 START:   LD      D,TRACK  ;Drive deseado a registro E
130          LD      C,&41    ;Núm. de pista a registro D
140          LD      HL,FTAB   ;Primer núm. sector en pista
150          LD      HL,FTAB   ;Tabla 32 bytes para FDC
160          RST    &18      ;Formato CALL &87
170          DEFW   FORMAT   ;Dir. de la dir. de 3 bytes FAR
180          RET
190 FORMAT:  DEFW   &C652   ;Dir. en AMSDOS de rutina &87
200          DEFB   7        ;Dir. precisa de ROM-select
210 FTAB:    EQU    $
220 SECT1:   DEFB   TRACK   ;Núm. de pista para sector-ID
230          DEFB   HEAD    ;Número de cabezal
240          DEFB   &41    ;Número del sector
250          DEFB   2        ;Magnitud sector para la ID
260 SECT2:   DEFB   TRACK
270          DEFB   HEAD
280          DEFB   &43
290          DEFB   2
300 SECT3:   DEFB   TRACK
310          DEFB   HEAD
320          DEFB   &45
330          DEFB   2
340 SECT4:   DEFB   TRACK
350          DEFB   HEAD
```

```
360      DEFB  &47
370      DEFB  2
380 SECT5: DEFB  TRACK
390      DEFB  HEAD
400      DEFB  &49
410      DEFB  2
420 SECT6: DEFB  TRACK
430      DEFB  HEAD
440      DEFB  &42
450      DEFB  2
460 SECT7: DEFB  TRACK
470      DEFB  HEAD
480      DEFB  &44
490      DEFB  2
500 SECT8: DEFB  TRACK
510      DEFB  HEAD
520      DEFB  &46
530      DEFB  2
540 SECT9: DEFB  TRACK
550      DEFB  HEAD
560      DEFB  &48
570      DEFB  2
```

2.2.2.7 La orden &87 Seek Track

Si bien todos los ejemplos con acceso directo al disco se dirigian hasta ahora sin mès a la pista deseada, para ciertas tareas puede tener pleno sentido posicionar el cabezal en una pista determinada. Se encarga de esta tarea la orden &87.

Para el posicionamiento sólo se requieren dos paràmetros. En primer lugar se pide la unidad deseada. Como segundo paràmetro, el nùmero de pista a la que ha de llegar el cabezal.

Tambièn en el caso de esta rutina, los programadores han aprovechado los registros como interface. El nùmero de

unidad es necesario en el registro E, el registro D ha de contener el número de pista.

También la utilización de flags como indicativos del éxito de la rutina es habitual. Un carry activado le notifica que la pista buscada ha sido hallada en el disco.

Dado que el programa necesario tampoco difiere esencialmente de los anteriormente impresos, en este caso no nos extenderemos más sobre ello.

2.2.2.8 La orden &88 Test Drive

¿Desea conocer desde un programa en código máquina si está disponible una determinada unidad de disco? Nada más sencillo. La orden &88 revela toda la información de interés sobre la unidad seleccionada. En cualquier caso presenta esta rutina una cierta peculiaridad, ya que la información sobre la unidad a comprobar no se halla como es usual en el registro E, sino en el acumulador.

Tras el retorno de esta rutina puede saberse, de los flags y del contenido del acumulador, si la unidad seleccionada está disponible. En caso de una ejecución de la rutina sin errores, el flag de carry está activado. En tal caso, el acumulador posee el contenido del registro de estado 0 del FDC. Lo único interesante para nosotros es que la disponibilidad de las unidades queda marcada en el acumulador mediante un cero (para la unidad A) o por un uno (para la unidad B).

2.2.2.9 La orden &89 Retry Count

Si el cabezal ha de ser posicionado en una determinada pista, ello puede lograrse fácilmente mediante la orden &87. Y es que con esta orden se le comunica al controlador la pista deseada.

Despuès de que la unidad de disco haya movido el cabezal de escritura/lectura el nùmero correspondiente de pasos en el sentido deseado, el FDC lee en la pista del disco, por si mismo, el nùmero de pista necesario en el formateado.

Si el nùmero leido coincide con el deseado, el comando ha llegado con éxito a su fin. Otro caso es cuando no puede leerse informaciòn alguna o bien, si puede leerse pero no coincide con el nùmero de pista deseado. En tal caso se lleva a cabo un nuevo intento de hallar la pista requerida.

Es aqui donde aparece la orden &89. Con ella resulta posible fijar el nùmero de intentos de lectura tras el posicionado. El AMSDOS instaura un valor de defecto de 10 intentos. Ello suele bastar, pero pudiera resultar necesario ampliar esta cifra.

El nuevo valor deseado es transferido a la rutina &89 a travès del acumulador. Pero en ello hay que tener en cuenta que el valor de 0 es interpretado como el valor màximo, de 256. No es imprescindible insertar valores menores que 10, pero con frecuentes intentos podemos echar a perder discos con problemas de lectura.

Para comprobar los efectos sin ensamblador pruebe a asignar mediante POKE el valor 0 a la posiciòn de memoria &BE66. Ello equivale a la llamada a la orden &89 y a la transferencia de 0 al acumulador. Si ahora inserta un disco virgen, es decir, no formateado, e intenta observar el directorio, oirà claramente còmo el cabezal va de un lado a otro del disco, para hallar la pista precisa. Si, por el contrario, asigna mediante POKE el valor 1 a la posiciòn de memoria &BE66, su unidad se tomarà bastante menos interès en hallar datos en el disco vacío.

Capítulo 3**Técnica del floppy y del disco****3.1 El controlador del floppy**

Mientras que en el CPC 464 la electrónica del interface al floppy se conecta a la carcasa del ordenador, en el CPC 664 y en el CPC 6128 ésta se halla junto a la primera unidad de discos, dentro de la carcasa del ordenador. Para el funcionamiento del interface, resulta relativamente indiferente el tipo de CPC que posea usted, ya que se diferencian sólo ligeramente. Puede ver los bloques de funciones globales en el diagrama de bloques de la figura 16.

El Floppy Disk Controller (FDC) integrado uPD 765 constituye el punto central de la tarjeta controladora. Este IC representa el interface entre las unidades de disco y el procesador del CPC. De hecho pueden montarse floppys sin incorporar un FDC, pero la elevada 'inteligencia propia' del FDC simplifica radicalmente la construcción. El gasto en hardware necesario, pero también la magnitud del software operativo a escribir se reduce drásticamente por la incorporación de un FDC. Un ejemplo lo esclarecerá.

La unidad de disco 1541 de la casa Commodore, conocida por muchos de ustedes como unidad para el Commodore C64, es una unidad de disco sin FDC incorporado. Independientemente de la escasa velocidad de transferencia de datos (que a usted, como poseedor de CPC, le provocaría, cuando menos, una sonrisa), limitada por la construcción, el gasto en hardware es, en esta unidad, claramente mayor que en el floppy del CPC. La electrónica digital del 1541 contiene un procesador propio, dos IC periféricos de 40 patillas y una gran cantidad de IC TTL distintos. ¡Un CPC 664 completo contiene un número comparable de componentes!

El software operativo para el 1541 viene a ser, con sus 16K, doble al del AMSDOS. Sin ninguna duda, tanto proyectistas (por comodidad) como comerciantes (por motivos de costes) prefieren el fácilmente incorporable FDC.

Antes de abrir, por así decirlo, la tapa del FDC en las próximas páginas, echémosle un vistazo al diagrama scrutando los distintos niveles.

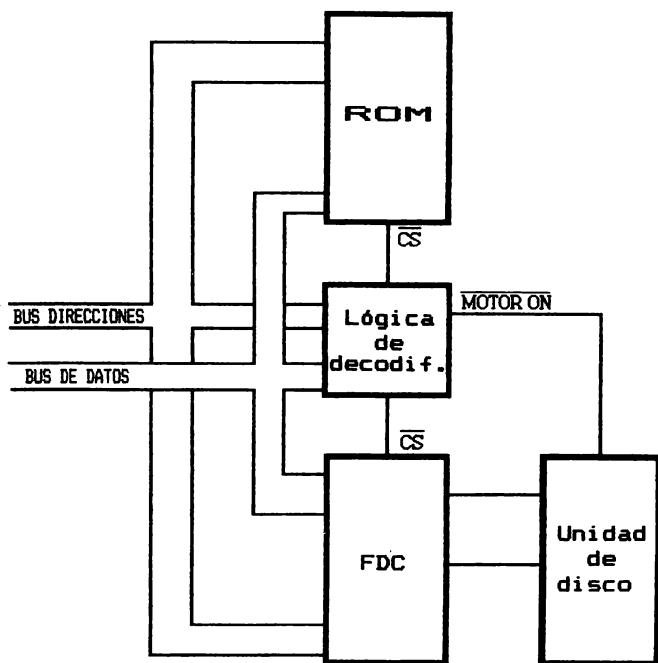


Figura 16

3.1.1 Descripción de la interfase del floppy

Como se puede observar en el diagrama de bloques de la figura 16, toda la electrónica del interface puede dividirse en tres grupos funcionales. El primer grupo funcional consta de la ROM del AMSDOS y de un flip-flop. El flip-flop tiene la misión de desconectar -bajo comando- el BASIC del CPC y activar la ROM del AMSDOS. Este flip-flop es activado cuando en la dirección de port &DFxx se da salida a un valor de &07. La ROM del AMSDOS está ahora activada en la zona de memoria desde &C000 a &FFFF en un acceso de lectura. Cualquier otro valor desconecta el flip-flop y, con ello, la ROM del AMSDOS.

Dado que la ROM se halla en un zócalo de IC de 28 patillas, resulta posible modificar el sistema operativo cambiándolo por una EPROM de 16 K sin meternos en chapuzas. Y en los 16 K utilizados sobra espacio para ampliaciones propias; el AMSDOS ocupa escasamente 8 K de la memoria disponible. Los 8 K restantes se precisan para el LOGO, sin lo que apenas habría espacio para programas LOGO en esta zona en la ROM. Pero si podemos prescindir del lenguaje de programación LOGO, un sistema operativo ampliado para floppy siempre puede alcanzar los 16 K.

El segundo flip-flop representado en el diagrama controla los motores de las unidades de disco. La entrada de datos del flip-flop se halla en el bit de datos 0. El impulso de CLK para el flip-flop se consigue, a través de varias puertas, de los bits de dirección A7, A8 y A10 así como de las señales IORQ y WR. Los bits de dirección mencionados han de estar a un nivel low para disparar el flip-flop. De ello resulta una dirección de port de &FA7x, en el AMSDOS el flip-flop se referencia a través del PORT &FA7E.

Las puertas para la decodificación de la dirección de port del flip-flop de motor se utilizan también para la decodificación de las direcciones del FDC. Pero en tal caso, el bit de dirección A8 ha de estar a nivel alto para generar la señal CS. Esto da como resultado la dirección de port

&FB7x. Dado que el bit de dirección del procesador A0 se utiliza para la elección de ambos registros del FDC, resultan las direcciones de port &FB7E para el registro principal de estado y &FB7F para el registro de datos.

También resulta interesante la generación de las dos señales de Drive Select (selección de unidad). Si bien el FDC dispone de dos conexiones separadas para la generación del Drive Select, sólo una de ellas se utiliza. La señal de salida de la patilla 29, USO, se utiliza como señal selectora conectada a través de una puerta NAND para la unidad B, y a través de una segunda puerta NAND para la unidad A. Según la polaridad de la señal USO, la unidad activa es A o B. Gracias a este diseño, los programadores del AMSDOS vieron facilitada su tarea. Basta con decrementar simplemente el valor para la elección de la unidad de disco, para poder determinar qué unidad ha de ser activada. Si el valor tras el decremento no es nulo, se activa la unidad A; si es nulo, se activa la unidad B.

3.1.2 El FDC 765

El FDC distribuido como uPD 765 por la firma NEC, como R 6765 por ROCKWELL y como 8765 por INTEL puede considerarse como un microprocesador muy especializado. Las posibilidades de este CI son tan amplias y complejas, que si la siguiente descripción peca por algo, no es por exceso.

El formato de datos utilizado por el FDC corresponde al formato 3740 de IBM en Single Density y al IBM System 84 en Double Density. Debido a esta limitación, lamentablemente no pueden leerse o escribirse discos de Commodore o Apple, por ejemplo.

Con sus cuarenta patillas ofrece todas las señales necesarias para el manejo de las unidades de disco usuales de las medidas de 8'', 5 1/4'' y 3''. Mediante las señales de control existentes, el diseñador está en disposición de conectar este FDC a casi cualquier procesador. Caben en ello dos posibilidades para la conexión y funcionamiento. El

primer mètodo es el funcionamiento DMA (Acceso Directo a Memoria). Junto con un controlador DAM, el FDC puede tomar el control de las funciones de transferencia de datos a travès de la memoria del sistema del ordenador en la lectura y escritura. En tal caso, con la ayuda del controlador DMA, recoge de memoria nuevos datos o escribe en memoria los valores leídos de disco, evitando siempre el procesador. Este es un mètodo muy ràpido de transferencia de datos pero no està incorporado al CPC, y se menciona aquí simplemente para dar una visiòn global.

En el segundo mètodo, el incorporado al CPC, la transferencia de datos se lleva acabo por el microprocesador. En dicho mètodo cabe distinguir a su vez entre dos posibilidades para el servicio del FDC.

En primer lugar està el mètodo de interrupts (interrupciones). Aquí se genera una interrupciòn para cada transferencia de datos. Tras ello ha de proporcionar o leer el procesador el siguiente byte de datos o de órdenes en la correspondiente rutina de interrupciòn del procesador. Debido a la construcciòn del hardware, esta posibilidad no pudo tenerse en cuenta, de modo que el proyectista utilizò el mètodo de "polling" o escrutinio. El procesador ha de mirar regularmente en los registros del FDC la acciòn que éste reclama como pròxima.

Observemos ahora los datos sobre las prestaciones del 765. Pero piense en que el fabricante de la placa controladora no ha agotado las posibilidades del 765.

- * Longitud de sector programable
- * Programabilidad de todos los datos de unidades
- * Posibilidad de conexiòn de hasta cuatro unidades
- * Transferencia de datos en modo DMA o no-DMA a elegir
- * Conectable a casi cualquier tipo usual de procesador
- * Alimentaciòn sencilla de 5 Voltios
- * Reloj sencillo de una fase de 4 u 8 MHz
- * Encapsulado de IC de 40 patillas

Ahora nos centraremos con más detalle en el último punto de esta breve exposiciòn.

3.1.2.1 La disposición de conexiones del FDC

Las conexiones del FDC pueden dividirse en distintos grupos. El primer grupo de conexiones supone el interface con el microprocesador. Por tanto, será a través de estas conexiones por las que se lleve a cabo el control del FDC por el procesador.

El segundo grupo sólo es preciso en relación con el funcionamiento en modo DMA. Por aquí se comunican el controlador DMA y el FDC.

El interface a los floppys es suministrado por el tercer grupo de conexiones, que con 19, es el grupo más numeroso.

En el cuarto y último grupo se reunen las conexiones para la alimentación y el reloj.

Comencemos con las conexiones del primer grupo, el interface con el procesador.

El interface con el procesador

RESET:

La entrada de RESET del FDC es activa ante un nivel alto (high). En el servicio normal, dicha conexión se halla conectada a masa. Con un nivel alto en la patilla de RESET, el FDC llega a un estado definido.

CS*: CHIP SELECT.

Mediante un nivel bajo (low) en esta patilla se selecciona el FDC. Previamente un CS*=low son válidos RD* y WR* para el FDC. Dado que la generación del CS queda en manos del fabricante, puede funcionar a elección como memory-mapped, es decir, llamado como parte de la memoria o a través de direcciones de port.

RD*: READ*

Esta conexión ha de ser unida a la señal RD* del microprocesador. Si éste desea leer datos del FDC esta línea ha de ser puesta a nivel bajo.

WR*: WRITE*

Así como la línea RD* delata accesos de lectura del procesador, un nivel bajo en WR* indica que el procesador está escribiendo datos u órdenes en el FDC.

A0*: ADDRESS LINE 0

El FDC sólo dispone de 2 direcciones accesibles desde el exterior. Su distinción se lleva a cabo mediante A0. Esta línea suele ir unida al bit de dirección más bajo del procesador.

DB0 - DB7: DATABUS 0-7

Estas conexiones del FDC están unidas al bus de datos del sistema. Todos los comandos y datos son transportados a través de estas ocho conexiones bidireccionales. El sentido correspondiente de transporte es fijado por el procesador o por el controlador DMA en el modo DMA.

INT: INTERRUPT

El FDC puede generar una interrupción en el procesador del sistema a través de esta conexión. Las interrupciones se generan para cada transferencia de un byte. (No se halla conectado en el CPC)

Señal es para el modo DMA (no utilizado en el CPC)

DRQ: DMA REQUEST

Por esta conexión indica el FDC al controlador DMA que ha de conseguir un acceso a memoria. En la siguiente oportunidad el controlador DMA toma el control del bus del sistema. El procesador se desconecta.

DACK*: DMA ACKNOWLEDGE

Indica que el controlador DMA se hizo cargo del bus y comenzó la transferencia de datos.

TC: TERMINAL COUNT

Con nivel alto en esta conexión se interrumpe la transferencia de datos, desde y al FDC. Si bien dicha conexión se usa generalmente en el modo DMA, también puede utilizarse para interrumpir transferencias en sistemas controlados por interrupciones.

El interface con el floppy**US0,US1: UNIT SELECT 0/1**

A través de estas conexiones pueden conectarse dos unidades de modo directo, o bien cuatro unidades con la ayuda de un decodificador dos-a-cuatro. A través de estas conexiones se selecciona la unidad elegida a fin de llevar a cabo una lectura o una escritura.

HD: HEAD SELECT

Dado que el FDC está preparado para el servicio de unidades con dos cabezales, puede elegirse a través de esta conexión el cabezal deseado.

HDL: HEAD LOAD

Esta se incorpora casi exclusivamente en unidades de 8". Los motores de estas unidades no se conectan cuando se necesita, sino que normalmente ya están continuamente en funcionamiento. Para proteger el disco y el cabezal de lectura/escritura, éste únicamente se pone sobre la superficie del disco en caso de necesidad. El electroimán elevador que se encarga de esto, puede ser controlado mediante HDL.

IDX: INDEX

En esta conexión se recoge el resultado de la acción del sensor óptico de índice y se le indica al FDC el inicio físico de una pista.

RDY: READY

La señal de READY enviada por el floppy muestra que en la unidad se halla un disco y que gira a una determinada velocidad mínima. Sólo tras la aparición del READY accede el FDC a la unidad.

WE: WRITE ENABLE

Esta salida del FDC ha de estar a nivel alto para poder escribir datos en el disco.

RW/SEEK: READ WRITE/SEEK

Una unidad de discos envía en total más señales de las que se hallan a disposición del interface con el floppy en un integrado de 40 patillas. Pero de todos modos, las señales no se necesitan todas a la vez. Por ello se han dividido 8 de ellas en dos grupos, que pueden ser dispuestas a voluntad en cuatro conexiones del FDC, que las elige por sí mismo a través de la conexión RW/SEEK.

FR/STP : FIT RESET/STEP

Esta es la primera de las cuatro señales dobles en el FDC. Esta salida tiene distintos significados según la operación realizada. En primer lugar puede volver a fijarse el error de flip-flop existente en algunas unidades. La segunda y con mucho la más frecuente utilidad es el comando de las entradas de impulso de paso de la unidad. Con cada cambio del cabezal se envían los impulsos necesarios a esta conexión.

FLT/TRO: FAUL/TRACK0

Esta entrada también puede presentar dos señales distintas. Si se realiza una operación SEEK (ver programación del FDC) se espera a la salida de esta conexión, una señal de track 0 de la unidad. Esta señal se genera mediante un sensor óptico o un conmutador mecánico cuando el cabezal de escritura/lectura se halla en la pista física 0. La segunda función, la señal de FAULT, es generada por algunas unidades de disco en caso de error y puede volver a ser borrada por el FDC mediante la señal FR/STP descrita anteriormente. Se comprueba esta señal en las operaciones de lectura/escritura del FDC.

LCT/DIR: LOW CURRENT/DIRECTION

Los impulsos de paso (step) de FR/STP indican únicamente que el cabezal ha de ser movido. LCT/DIR especifica en el modo SEEK el sentido del movimiento del cabezal. La función LOW CURRENT es necesaria en la escritura de los datos. Mediante esta señal puede limitarse la corriente de escritura a las pistas interiores. Hallará sus particularidades en las bases teóricas del almacenamiento en disco.

WP/TS: WRITE PROTECT/TWO SIDE

En todas las unidades de disco, el estado de la protección de escritura es notificado al controlador como señal por parte de la unidad. Esta señal se comprueba en la entrada WP/TS en operaciones de lectura/escritura. La señal TS es comprobada en operaciones SEEK. Sólo se necesita en unidades de dos cabezales.

WDA: WRITE DATA

Por esta conexión se proporcionan a la unidad en serie los datos de escritura. Puede tratarse tanto de los datos que se presentan en la escritura de un sector como de toda la información precisa para formatear.

PS0,1: PRE SHIFT 0/1

A través de esta conexión comunica el FDC, ante un formato de doble densidad (MFM), a una parte electrónica adecuada cómo ha de ser escrito sobre el disco el flujo de datos. Para la precompensación caben estos tres estados: EARLY, NORMAL y LATE.

RD: READ DATA

Por esta conexión se proporcionan al FDC los datos leídos del disco. A partir de este flujo serial de bits se recuperan los bytes escritos.

RDW: READ DATA WINDOW

Esta señal se consigue en un separador de datos a partir de los datos leídos. Hallará más sobre el asunto en el capítulo "Fundamentos del Almacenamiento en Disco".

VCO: VCO SYN

Esta señal es precisa en el separador de datos PLL para el control del VCO.

MFM: MFM MODE

Esta conexión señala si el controlador está trabajando en el formato simple densidad o bien en el formato doble densidad.

Alimentacin y seales de reloj**Vcc: +5 V**

A travs de esta conexin obtiene el FDC su tensin de alimentacin. Esta tensin de 5 V debera ser constante con una tolerancia del +- 5%. La intensidad requerida supone como mximo unos 150 mA.

GND: GROUND**Conexin de masa del FDC.****CLK: CLOCK**

El FDC precisa una frecuencia de que segn la unidad de disco, ha de ser de 4 MHz (en las de 5 1/4" y menores), o de 8 MHz (en 8").

WCK: WRITE CLOCK

La frecuencia de esta seal ha de elejirse de acuerdo con el formato. En MF, la frecuencia ha de ser de 500 kHz y en MFM ha de ser de 1 Mhz. Esta frecuencia determina la velocidad de transferencia de los datos de y hacia el floppy.

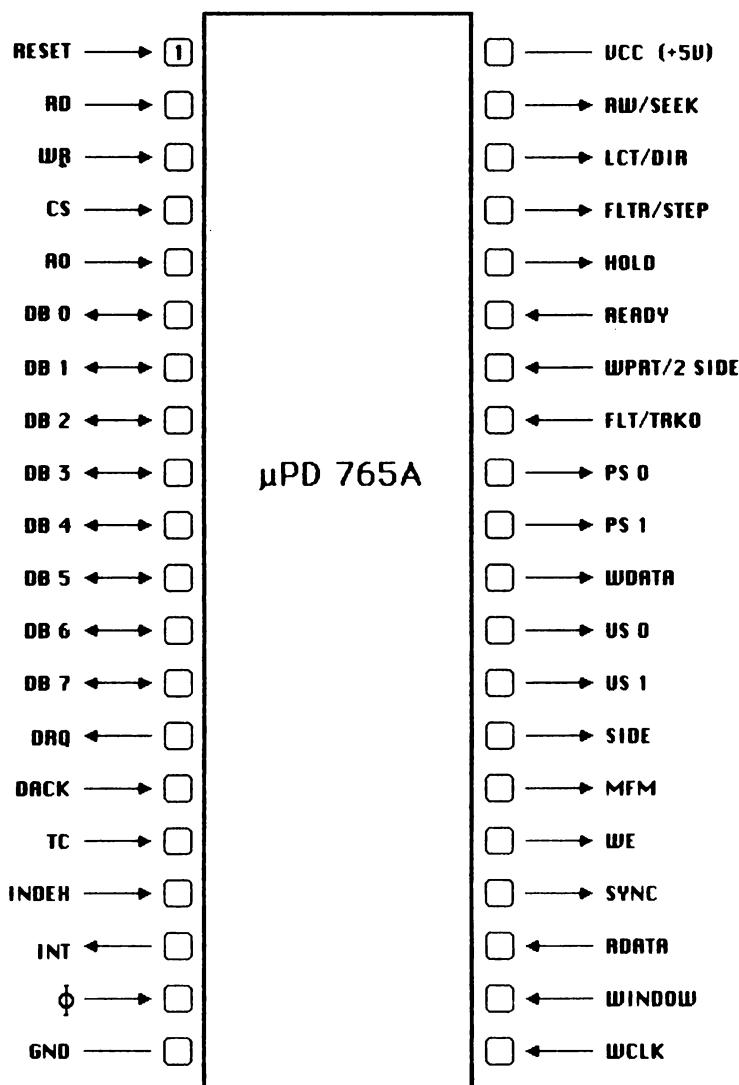


Figura 17.

3.1.2.2 La programaciòn del FDC 765

El FDC 765 dispone en el exterior de sólo dos direcciones o registros. El registro disponible queda fijado por el nivel de la señal A0. Si A0 se halla a potencial de masa, puede accederse al registro principal de estado. Este registro principal de estado sólo puede ser leido. Un nivel alto en A0 permite el acceso al registro de datos. El registro de datos puede ser tanto escrito como leido. A través de este registro se programa el FDC, se transfieren todos los datos de escritura y lectura y se envian al procesador los datos de la fase de resultados.

En los comandos del FDC cabe distinguir fundamentalmente tres fases diferenciadas. La primera es la fase de comando o de orden. En esta fase se transfieren al FDC todos los paràmetros necesarios para el comando. Ello puede suponer hasta nueve bytes en algunos comandos. Una vez notificados al FDC todos los bytes de esta fase, comienza la fase de ejecuciòn (execute). El significado concreto es que, por ejemplo, los datos llegan ahora, tras una orden de lectura. Finalizada la fase de ejecuciòn, comienza la òltima, la fase de resultados (result). Durante la fase de resultados proporciona el FDC un total de hasta siete informaciones de estado, que han de ser leídas por el procesador.

En cualquier caso, el esquema mostrado no afecta a toda orden. En algunas órdenes no existe fase de resultados, otras órdenes no tienen fase de ejecuciòn. Una orden llega a prescindir incluso de ambas. Le basta la fase de órdenes. Antes de comentar cada uno de los comandos, demos un vistazo a las informaciones de estado.

El FDC 765 dispone de un total de cinco registros de estado. El registro principal de estado ocupa, como ya mencionamos, una dirección propia (A0 = low), y, como todos los demás registros de estado, sólo puede ser leido. De todos modos, siempre resulta posible un acceso a dicho registro, incluso durante la elaboraciòn de una orden.

Los otros cuatro registros de estado sólo son accesibles para algunas órdenes en la fase de resultados. El primer byte de dichas órdenes muestra en la fase de resultados el estado del registro de estado 0 (ST0). Los estados de ST1 y ST2 se envían detrás, en la fase de resultados, como bytes dos y tres. Solamente obtendremos el estado de ST3, el último registro de estado, si lo pedimos mediante una orden especial que tiene por único objetivo proporcionarnos dicho estado. El significado del registro de estado es algo que aprenderemos más adelante.

El FDC 765 dispone en total de 15 comandos diferentes. A la vista de este número ya se ve que este controlador tiene más posibilidades que la simple lectura y escritura. Ahora se trata de ver detalladamente qué posibilidades ofrece. Si algunos de los conceptos no explicados en este capítulo, como por ejemplo, sector ID, Gap o Data Adress Mark, no le dicen nada, puede revisarlos en el capítulo sobre la representación física de los datos.

Leer datos

Antes de que el FDC pueda leer datos del disco, han de serle transferidos 9 bytes en la fase de órdenes. Tras dar todos los datos se activa la señal Head Load y se espera el tiempo de Head Load programado. Tras ello lee el FDC IDs (identificadores) de sector hasta que halla el ID del sector dado o que el impulso de índice aparezca por segunda vez tras el comienzo de la búsqueda. En el primer caso comienza con la fase de ejecución, en el segundo caso finaliza el comando y comienza la fase de resultados.

Tan pronto ha identificado un sector, comienza con la fase de ejecución. En esta fase se leen los datos del disco y envían al procesador a través del bus de datos. Los intervalos de tiempo en que esto ocurre son muy cortos. Aproximadamente cada 26 microsegundos hay un byte disponible que ha de ser leído por el procesador.

Tras la transferencia del último byte del sector deseado,

hay que darle al FDC un impulso de TERMINAL COUNT (TC, patilla 16), para encarrilar la fase de resultados, ya que sin el impulso de TC se llevaria a cabo un denominado multi-sector-read (lectura múltiple de sectores).

Multi-sector-read significa que el controlador lee datos hasta el último sector de la pista. De todos modos, bajo ciertas circunstancias, puede prescindirse del impulso TC. En la fase de sector de 'leer sector' y en algún comando más, ha de darse no sólo el sector, deseado sino también el último de la pista. Si ambos valores son idénticos, el FDC interrumpe la lectura automáticamente al final del sector deseado y comienza con la fase de resultados.

MT Multitrack-bit: si está activado, la función multi-sector sigue en la segunda cara del disco; sólo disponible en unidades dobles, siempre 0 en el AMSDOS.

MF MFM-mode-bit: si está activado, el FDC trabaja en doble densidad (double Density), siempre 1 en el AMSDOS.

SK Skip-bit: si está activado, salta sectores borrados. No incorporable bajo AMSDOS y CP/M. Siempre 0.

HD Head-select-bit: selecciona cara en unidades dobles. En el AMSDOS siempre es 0.

US 0,1 Unit select: elige unidad. En AMSDOS 0 para unidad A, 1 para unidad B.

R Read: lectura

W Write: escritura

Estos indicativos y abreviaturas son igualmente válidos para todas las representaciones que siguen.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
LEER DATOS										
Comando	W	MT	MF	SK	0	0	1	1	0	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	
	W	----- número de pista -----								Información sector ID antes de ejecutar el comando
	W	----- dirección de cabezal -----								comando
	W	----- dirección de sector -----								
	W	----- longitud de sector -----								
	W	- último nr. sector en pista -								
	W	espacio entre datos ID y long.								
	W	long. datos si long. sector =0								
Ejecución										Transferencia de datos entre FDD y sistema
Resultado	R	----- estado 0 -----								Informe de estado tras ejecución del comando
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Información sector ID tras ejecución del comando
	R	----- dirección cabezal -----								
	R	----- dirección sector -----								
	R	----- longitud sector -----								

Figura 18

En la fase de resultados el FDC proporciona 7 bytes al procesador, que, a la mayor brevedad, deberà recoger, es decir, leerlos. El FDC no acepta ningún comando nuevo en tanto en cuanto no haya sido leido el último byte de la fase de resultados.

Los tres primeros bytes proporcionados son los estados 0 a 2 del registro de estado. El programador puede leer todos los datos relevantes sobre el éxito o fracaso de la orden, de la mano de estos tres registros.

Ademàs se proporcionan el número actual de pista, la dirección de cabezal (importante en unidades de dos cabezales), el número de sector y la longitud de sector. Sólo tras ello puede proseguirse con un nuevo comando.

Escribir datos

Igual que en la lectura, en la escritura de sectores son necesarios 9 bytes. Tras haber transferido todos los bytes al FDC, éste comienza (transcurrido el Head Load) la búsqueda del sector deseado. Para ello lee IDs de sector hasta que identifique el sector deseado o bien aparezca por segunda vez en la búsqueda el impulso de índice. En el segundo caso, se interrumpe el comando e inmediatamente después comienza la fase de resultados.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
ESCRIBIR DATOS										
Comando	W	MT	MF	0	0	0	1	0	1	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	
Ejecución	W	----- número de pista -----								Información sector ID antes de ejecución de comando
	W	----- dirección de cabezal -----								
	W	----- dirección de sector -----								
	W	----- longitud de sector -----								
	W	- último nr. sector en pista -								
	W	espacio entre datos ID y long.								
	W	long. datos si long. sector =0								
Resultado	R	----- estado 0 -----								Transferencia de datos entre FDD <=> sistema
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Informe de estado tras ejecución
	R	----- dirección cabezal -----								
	R	----- dirección sector -----								
	R	----- longitud sector -----								Información sector tras ejecución del comando

Figura 19

Pero si se halla el sector deseado, el FDC requiere del procesador (en el servicio de no-DMA) los datos a escribir. Tras haber sido escritos todos los datos, comienza la fase de resultados. No varía con la del comando leer datos.

Ler datos borrados

El indicativo para esta orden es algo confuso. No han de confundirse datos borrados con los datos de un fichero borrado. El FDC no tiene ni idea de ficheros ni directorios. 'Borrados' se refiere a la posibilidad de marcar como borrados a sectores mediante la entrada de un Data Address Mark borrado. Tales sectores son ignorados en una lectura/escritura normal. Por lo demás, la lectura de datos borrados no difiere de la normal

Fase	R/W	BUS DE DATOS								Observaciones										
		D7	D6	D5	D4	D3	D2	D1	D0											
LEER DATOS BORRADOS																				
Comando	W	MT	MF	SK	0	1	1	0	0	Códigos de comando ----- número de pista ----- ----- dirección de cabezal ----- ----- dirección de sector ----- ----- longitud de sector ----- ----- último nr. sector en pista ----- espacio entre datos ID y long. long. datos si long. sector =0										
	W	X	X	X	X	X	HD	US1	US0											
	W									Información sector previa a ejecución del comando										
	W																			
	W																			
	W																			
	W																			
Ejecución																				
Resultado	R	----- estado 0 -----								Transferencia de datos FDD <=> sistema Informe de estado tras ejecución Información sector ID tras ejecución del comando										
	R	----- estado 1 -----																		
	R	----- estado 2 -----																		
	R	----- número de pista -----																		
	R	----- dirección cabezal -----																		
	R	----- dirección sector -----																		
	R	----- longitud sector -----																		

Figura 20

Escribir datos borrados

Esta orden no se diferencia en esencia de la escritura normal de datos. La única excepción la constituye el tratamiento especial de la Data Adress Mark; aquí se registra una Data Adress Mark borrada.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
LEER DATOS BORRADOS										
Comando	W	MT	MF	0	0	1	0	0	1	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	
	W	----- número de pista -----								Información sector ID antes de la ejecución del comando
	W	----- dirección de cabezal -----								
	W	----- dirección de sector -----								
	W	----- longitud de sector -----								
	W	- último nr. sector en pista - espacio entre datos ID y long. long. datos si long. sector =0								
Ejecución	R	Transferencia de datos FDD <=> sistema								
	R	----- estado 0 -----								Informe de estado tras ejecución
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Información sector ID tras ejecución del comando
	R	----- dirección cabezal -----								
	R	----- dirección sector -----								
Resultado	R	----- longitud sector -----								

Figura 21

Lectura de una pista

Este comando es comparable a la lectura de un sector. La única particularidad es la de que con él se leen todos los bytes de datos de la pista. Este comando finaliza cuando haya sido leído el sector indicado como último o, si éste no fue hallado, cuando la perforación de índice genere el segundo impulso tras el inicio del comando.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
LEER DATOS BORRADOS										
Comando	W	0	MF	SK	0	0	0	1	0	Códigos de comando ----- número de pista ----- ----- dirección de cabezal ----- ----- dirección de sector ----- ----- longitud de sector ----- ----- último nr. sector en pista ----- espacio entre datos ID y long. long. datos si long. sector =0
	W	X	X	X	X	X	HD	US1	US0	
	W									
	W									
	W									
	W									
Ejecución										Información sector ID antes de la ejecución del comando
										Transferencia de datos en- tre FDD y el sistema. El FDC lee todo lo que está en la pista entre la perfo- ración de índice y EOT.
Resultado	R									Informe de estado tras ejecución
	R									Información sector ID tras ejecución del comando
	R									
	R									
	R									
	R									

Figura 22

Formatear una pista

El formateo de una pista resulta muy sencillo con el 765. Para ello se transfiere al FDC una cadena de 6 bytes en la fase de órdenes. Si el FDC reconoce en el impulso de índice el inicio físico de la pista tras la transferencia completa de los seis bytes, comienza automáticamente a formatear la pista con todas las Address Marks, Gaps e IDs precisas.

El byte 4 de la cadena de orden indica cuantos sectores hay que instalar en cada pista. El FDC requiere cuatro bytes más por sector. Uno es el número de sector, que queda anotado en el sector ID. Esto permite formatear sectores en distintos órdenes. Con ello puede, mediante hábil elección del orden, acelerar sustancialmente los posteriores accesos a lectura.

Fase	R/W	BUS DE DATOS								Observaciones	
		D7	D6	D5	D4	D3	D2	D1	D0		
FORMATEAR PISTA											
Comando	W	0	MF	0	0	1	1	0	1	Códigos de comando Bytes/sector Sectores/pista Byte de relleno FDC formatea la pista completa	
	W	X	X	X	X	X	HD	US1	US0		
	W	----- longitud de sector -----									
	W	----- sectores por pista -----									
Ejecución	W	---espacio entre ID y datos---								Byte de relleno FDC formatea la pista completa	
	W	-modelo de datos para sector-									
	R	----- estado 0 -----								Informe de estado tras ejecución	
	R	----- estado 1 -----									
Resultado	R	----- estado 2 -----								En este caso la información no tiene ningún significado	
	R	----- número de pista -----									
	R	----- dirección cabezal -----									
	R	----- número sector -----									
----- longitud sector -----											

Figura 23.

Leer ID

Con este comando puede leer del disco el siguiente ID posible, dando previamente dos bytes en la fase de órdenes. Cada sector obtuvo su propio ID en el formateado. En este ID se hallan los valores del número de sector, de pista, de cara del disco y de la longitud de sector. Estos valores son transferidos al procesador en la fase final de resultados junto con los estados de los tres registros de estado ST0 a ST2.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
LEER ID										
Comando ¹	W	0	MF	0	0	1	0	1	0	Códigos de comando
Ejecución	W	X	X	X	X	X	HD	US1	US0	Almacenar en el registro de datos la primera información correcta de ID de una pista
Resultado	R	----- estado 0 -----								Informe de estado tras ejecución
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Información sector ID antes de la ejecución del comando
	R	----- dirección cabezal -----								
	R	----- número sector -----								
	R	----- longitud sector -----								

Figura 24.

Los comandos de scan, comprobación de un sector

Estas órdenes tienen un efecto de verificación(verify), es decir, de comparación entre los datos escritos y a escribir. Las condiciones de prueba son 'igualdad', 'mayor o igual que' y 'menor o igual que'. El FDC lee datos del sector elegido tras la transferencia de 9 bytes en la fase de órdenes. El FDC reclama simultáneamente datos al procesador. Cada byte del disco es comparado con un byte del procesador siguiendo la condición de prueba dada. Este comando finaliza cuando la condición de prueba se cumple en todo el sector dado, o bien se comprueba el último sector de la pista, o bien si se produjo un impulso TC en la patilla 16.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
SCAN IGUAL										
Comando	W	MT	MF	SK	1	0	0	0	1	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	
Ejecución	W	----- número de pista -----								Información sector ID antes de la ejecución del comando
	W	----- dirección de cabezal -----								
	W	----- dirección de sector -----								
	W	----- longitud de sector -----								
	W	----- último n. sector en pista -----								
	W	----- espacio entre datos ID y long. -----								
	W	----- long. datos si long. sector =0 -----								
Resultado	R	----- estado 0 -----								Transferencia de datos FDD <=> sistema
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Informe de estado tras ejecución
	R	----- dirección cabezal -----								
	R	----- dirección sector -----								Información sector ID tras ejecución del comando
	R	----- longitud sector -----								

Figura 25.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
SCAN MENOR O IGUAL										
Comando	W	MT	MF	SK	1	1	0	0	1	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	Información sector ID antes de la ejecución del comando
	W	----- número de pista -----								
	W	----- dirección de cabezal -----								
	W	----- dirección de sector -----								
	W	----- longitud de sector -----								
	W	- último n. sector en pista - espacio entre datos ID y long. long. datos si long. sector =0								
Ejecución	W									
	R	----- estado 0 -----								Transferencia de datos FDD <=> sistema
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Informe de estado tras ejecución
	R	----- dirección cabezal -----								
	R	----- dirección sector -----								Información sector ID tras ejecución del comando
Resultado	R	----- longitud sector -----								

Figura 26.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
SCAN MAYOR O IGUAL										
Comando	W	MT	MF	SK	1	1	1	0	1	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	Información sector ID antes de la ejecución del comando
	W	----- número de pista -----								
	W	----- dirección de cabezal -----								
	W	----- dirección de sector -----								
	W	----- longitud de sector -----								
	W	- último n. sector en pista - espacio entre datos ID y long. long. datos si long. sector =0								
Ejecución	W									
	R	----- estado 0 -----								Transferencia de datos FDD <=> sistema
	R	----- estado 1 -----								
	R	----- estado 2 -----								
	R	----- número de pista -----								Informe de estado tras ejecución
	R	----- dirección cabezal -----								
	R	----- dirección sector -----								Información sector ID tras ejecución del comando
Resultado	R	----- longitud sector -----								

Figura 27.

Recalibrado, buscar pista cero.

Con este comando se mueve el cabezal de la unidad elegida hacia la pista cero hasta que, o bien el impulso de Track 0 indique al FDC que se ha alcanzado la pista, o bien el FDC haya suministrado ya 77 impulsos de paso. Puede determinarse cómo finalizó el comando mediante el registro de estado.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
BUSCAR PISTA 0										
Comando	W	0	0	0	0	0	1	1	1	Códigos de comando
	W	X	X	X	X	X	HD	US1	US0	
Ejecución										Cabez.a pista 0

Figura 28.

El comando SEEK, búsqueda de una pista

Las unidades de disco mueven el cabezal de escritura/lectura con ayuda de motores paso a paso. Estos motores no están en constante movimiento, sino que son movidos, mediante impulsos, en variaciones angulares muy definidas alrededor del eje. La generación de estos impulsos suele llevarse a cabo en la propia unidad por circuitos digitales relativamente manejables. Una unidad dispone de dos conexiones hacia el exterior. En una de ellas se disponen los impulsos que controlan el motor paso a paso de tal modo que el cabezal es desplazado exactamente en una pista con cada aparición de un impulso. La segunda entrada de las unidades determina el sentido deseado.

Este movimiento del cabezal se controla mediante el comando Seek. El FDC dispone en total de cuatro registros internos, en los que almacena la posición actual del cabezal de las cuatro posibles unidades. Estos cuatro registros se hallan tras el comando de recalibrado a cero, es decir, han sido

borrados. Si se envía un comando Seek a una determinada unidad, el contenido del registro de posición correspondiente es comparado con el valor requerido. Si los dos valores son iguales, no se requiere acción alguna. Pero si existe alguna diferencia entre ambos valores, se influencia la polaridad de la señal DIR en la patilla 38 correspondientemente al sentido preciso, y se generan impulsos de paso en la patilla 37. Puede programarse el intervalo entre dichos impulsos en un amplio rango mediante el comando 'entrar datos unidad'.

Ni en esta orden ni en el comando recalibrado existe fase de resultados. Tras un comando de Seek, el programador debería incluir siempre el comando 'leer estado de interrupciones', para finalizar el comando de un modo ortodoxo. Este comando proporciona en la fase de resultados el estado del STO, también llamado estado de interrupciones. Sin este comando, el FDC no vuelve a aceptar órdenes de escritura/lectura.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
BUSCAR PISTA										
Comando	W	0	0	0	0	1	1	1	1	Códigos de comando
Ejecución	W	X	X	X	X	X	HD	US1	US0	Posicionado del cabezal en pista deseada
	W	número de pista								

Figura 29.

Leer estado de interrupciones, registro de estado 0

El FDC genera interrupciones en el funcionamiento NO-DMA en las siguientes circunstancias:

- durante la fase de ejecución
- al comienzo de la fase de resultados
- al final de un Seek o recalibrado
- ante variación de la señal READY de una de las unidades

Si bien las dos primeras causas de interrupción pueden ser fácilmente reconocidas por el procesador, en las dos últimas ha de realizarse el comando 'leer estado interrupciones' para reconocer la causa. El origen de la interrupción puede hallarse fácilmente en los bits del valor del ST0 proporcionado.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
LEER ESTADO DE INTERRUPCIONES										
Comando	W	0	0	0	0	0	0	0	0	Códigos de comando
Resultado	R	----- estado 0 -----								Informe estado al final de operación búsq. por FDC
R		núm. pista tras orden búsqueda								

Figura 30.

Lee estado de unidad, registro de estado 3

Esta orden constituye la única posibilidad de conseguir el contenido del registro de estado ST3. Este registro informa acerca del estado de la unidad elegida y puede ser leído en cualquier momento mediante el correspondiente comando.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
LEER ESTADO UNIDAD										
Comando	W	0	0	0	0	0	1	0	0	Códigos de comando
	W	x	x	x	x	x	HD	US1	US0	
Resultado	R	----- estado 3 -----								Informe estado FDD

Figura 31.

El comando Specify, dar datos de unidad

Si bien està en la última posición en nuestra breve exposición de las órdenes del FDC, esta orden debería ser la primera tras un Reset o la conexión del FDC. Con ayuda de esta orden pueden ajustarse al FDC las unidades de disco más dispares. Todos los tiempos que se puedan plantear son notificados al FDC mediante este comando de tres bytes. Junto a los tiempos de demora, se decide si el FDC ha de trabajar en el modo DMA o en el modo de interrupción.

Pero ocupémonos de los datos de unidad. En primer lugar el llamado step rate time. Es el tiempo que aguarda por sí mismo el FDC entre dos impulsos de (step) de paso. Dado que las unidades de los distintos fabricantes suelen esperar tiempos totalmente diferentes entre los impulsos, puede ajustarse dicho tiempo exactamente a los valores precisos.

El segundo tiempo que podemos fijar es el tiempo de demora que el FDC introduce automáticamente tras activar la señal Head Load. Este tiempo de demora llamado Head Load Time sólo es relevante en unidades de 8", en unidades menores casi siempre se carga el cabezal con la señal Motor On.

El tercer tiempo que podemos fijar es el HeadUnload Time. Este tiempo programable es esperado tras el acceso a disco hasta que la señal Head Load del FDC vuelva a ser inactiva. También aquí, resulta únicamente significativo para unidades que efectivamente lleven a cabo el control del cabezal.

través de la mencionada conexión del FDC.

Fase	R/W	BUS DE DATOS								Observaciones
		D7	D6	D5	D4	D3	D2	D1	D0	
DAR DATOS UNIDAD										
Comando	W	0	0	0	0	0	0	1	1	Códigos de comando
	W	step rate-->--elevar cabezal								Informe estado final operación búsqueda por FDC
	W	cargar cabezal-----> no DMA								

step rate en 5 1/4"					
bit	7	6	5	4	tiempo
	0	0	0	0	32 ms
	0	0	0	1	30 ms
	0	0	1	0	28 ms
:	:				:
:	:				:
1	1	0	1		6 ms
1	1	1	0		4 ms
1	1	1	1		2 ms

cargar cabezal								
bit	7	6	5	4	3	2	1	tiempo
	0	0	0	0	0	0	0	4 ms
	0	0	0	0	0	0	1	8 ms
	0	0	0	0	0	1	0	12 ms
:	:							:
:	:							:
1	1	1	1	1	1	1	0	500 ms
1	1	1	1	1	1	1	0	504 ms
1	1	1	1	1	1	1	1	508 ms

elevar cabezal					
bit	3	2	1	0	tiempo
	0	0	0	0	0 ms
	0	0	0	1	32 ms
	0	0	1	0	64 ms
:	:				:
:	:				:
1	1	0	1		416 ms
1	1	1	0		448 ms
1	1	1	1		480 ms

DMA-bit							
1	= modo DMA						
0	= modo de NO DMA						

Figura 32.

3.1.2.3 Los registros de estado del FDC 765

Como ya comentamos, el 765 dispone de 5 registros internos de estado. El registro de estado principal es legible en cualquier momento. El contenido de los registros de estado 0 a 2 se obtiene en la conclusión de todas las órdenes de escritura y de lectura. El acceso dirigido a los registros 1 y 2 no resulta posible. Tan sólo puede leerse mediante acceso dirigido el contenido de los registros de estado 0 y 3.

La nomenclatura especial y las abreviaturas empleadas en la siguiente descripción están tomadas del folleto de NEC del 765. Lamentablemente, esta nomenclatura no ha sido utilizada constantemente, ni siquiera por NEC. De ahí que en el área de aplicaciones del 765 aparezcan algunas abreviaturas o una nomenclatura parcialmente distinta a la utilizada en el folleto.

Antes de ocuparnos en detalle de los registros de estado, hemos de explicar un concepto utilizado. El término "cylinder" (cilindro) no significa otra cosa que track, pista. El porqué de la aparición paralela de ambas denominaciones en el folleto resulta incomprensible para nosotros. Hemos intentado evitar en lo posible el concepto cylinder. Pero en algunos puntos, las abreviaciones empleadas perderían totalmente el sentido. Aquí se conservó el concepto cylinder.

El registro principal de estado

En este registro se representan los datos más importantes sobre el estado actual del FDC. También se regula mediante este registro todo el protocolo entre el procesador y el FDC. Los ocho bits de este registro dan los siguientes datos y estados:

Bit 7 RQM:	Request For Master
	Si este byte está activado, el FDC está listo para leer o enviar un nuevo byte a través del registro de datos. Por el contrario, con el RQM desactivado es imposible la transferencia.

- Bit 6 DIO:** Data Input/Output (Entrada/Salida de datos)
Si RQM muestra que una transferencia resulta posible, DIO indica el sentido preciso. Un DIO activado muestra que el FDC tiene un byte para el procesador. Un DIO desactivado indica que el FDC espera del procesador un byte.
- Bit 5 EXM:** Execution Mode (Modo de Ejecución)
Este bit sólo se utiliza en el modo DMA. En el funcionamiento DMA este bit suele estar desactivado. En el funcionamiento de NO-DMA el bit de EXM está activado si empezó la fase de ejecución. Con la ayuda de este bit puede, por tanto, determinarse si se trata de valores suministrados para informes de sector o de los bytes de la fase de resultados.
- Bit 4 CB:** FDC Busy (FAC ocupado)
Un bit de CB activado muestra que el FDC está elaborando una orden de escritura o lectura, y no puede encargarse de más comandos. Este bit se activa a la recepción del primer byte de una cadena de comando y permanece activado hasta la lectura del último byte de la fase de resultados. Tras ello, el bit es automáticamente desactivado.
- Bits 3-0 DB:** FDD3-0 Busy (Unidad ocupada)
Estos cuatro bits están asignados a las cuatro unidades posibles. Si en una de estas unidades se inicia un comando SEEK o recalibrado, el bit correspondiente es activado en el registro principal de estado. Cuando uno de estos bits está activado, no puede enviarse al FDC orden de escritura o lectura alguna. Pero siguen siendo posibles otras órdenes de SEEK o de recalibrado en las restantes unidades. En tal caso se activa el bit correspondiente para cada nueva orden.

Estos bits no se desactivan automáticamente al final de la orden. Para volver a poner estos bits a 0 ha de ser enviado al FDC el comando 'leer estado de interrupción'. Esta orden borra los bits, si finalizó el comando.

El registro de estado 0

El registro de estado 0 se designa como registro de estado de interrupciones, dado que en el funcionamiento de NO-DMA proporciona la causa de la interrupción.

Bit 6,7: **Interrupt Code (Código de interrupción)**
En estos dos bits el FDC aclara el transcurso de un comando. Caben cuatro posibilidades con estos dos bits:

Bit
7 6 .
0 0 Comando finalizado con éxito. Este sería el mensaje deseable, dado que significa que, p.ej, un acceso a lectura fue fructífero. Pero, ¡ojo!

0 1 Comando interrumpido. En este caso el comando fue comenzado, pero no finalizó con éxito. Tal mensaje puede aparecer en el caso de, p.ej, errores de lectura del disco, y en CPC surge también tras cada lectura o escritura de un sector. La causa es la falta de la señal TC y la consecuentemente necesaria programación del último sector de pista como el último sector a leer. Por ello, la aparición de esta condición no ha de entenderse necesariamente como un auténtico error.

1 0 Comando inválido. El comando dado no pudo comenzarse ya que se trataba de un comando ilegal.

También obtendrá este comando si ordenó 'Leer estado de interrupciones' sin que existiera, en aquel momento, interrupción alguna.

1 1 Comando interrumpido. La causa de este mensaje es una variación de la señal READY de la unidad elegida durante un comando. Y es que el comando comenzó, pero no fue completamente finalizado. Obtendrá este mensaje, p.ej, en caso de que retire el disco de la unidad durante un acceso a lectura.

Bit 5 SE: Seek End (Fin de Seek)
Tan pronto se finalice un comando de Seek, el FDC pone este bit a uno.

Bit 4 EC: Equipment Check (Error de equipo)
Este bit de estado muestra en primer lugar si la unidad notifica un error. En caso de error se activa el bit de EC. La segunda causa de un bit de EC activado es la permanencia de la TRKO señal tras un recalibrado. En el recalibrado se mueve el cabezal hacia la pista 0 hasta que o bien el sensor de pista 0 envíe un mensaje al FDC o bien hasta que hayan sido enviados 77 impulsos de paso a la unidad. Tal caso puede aparecer en unidades de 80 pistas cuando el cabezal de escritura/lectura se halle en las pistas de la 78 a la 80. Entonces puede repetirse el comando de Recalibrado

- Bit 3 NR:** Not Ready (No preparado)
Si la unidad elegida notifica en una orden de escritura o de lectura que no está lista, se activa este flag. También lo activa el acceso a un segundo cabezal no existente.
- Bit 2 HD:** Head Adress (Dirección de cabezal)
Este bit informa sobre el cabezal elegido en este momento de la interrupción.
- Bit 1,0 US:** Unit Select (Unidad activa)
Estos dos bits indican qué unidad está activa
en el momento de la interrupción.

El registro de estado 1

Este registro informa, en la fase de resultados, sobre el desarrollo de la fase de ejecución.

- Bit 7 EN:** End of Track (Fin de pista)
Este flag es activado por el FDC cuando intenta un acceso a un sector tras el fin de pista programado.
- Bit 6** no utilizado, siempre cero
- Bit 5 DE:** Data Error (Error en datos)
El FDC genera en la escritura de datos, y de modo automático, una suma de comprobación según el principio del 'Cyclic Redundancy Check (CRC)', que es almacenada en el disco junto a los datos. Estas sumas de comprobación (checksum) se forman de igual modo en la lectura de datos y se comparan con los valores almacenados. Si el FDC halla alguna diferencia entre ambas sumas de comprobación, la de los campos de datos y la de los campos de ID, el flag de DE se activa.

- Bit 4 OR: Over Run (Se excedió tiempo de transferencia)
La transferencia de datos entre el procesador y el FDC ha de llevarse a cabo en la lectura o escritura de los datos antes de un máximo de tiempo. De este modo, los datos son enviados al procesador a intervalos de sólo 26 us. Si el procesador no puede soportar por alguna causa esta velocidad, ocurre que hay un nuevo byte preparado para la lectura, antes de que el procesador haya tenido tiempo de leer el anterior. En tal caso se habla de Over Run, y el bit de OR se activa.
- Bit 3 no utilizado, siempre cero
- Bit 2 ND: No Data
La activación de este flag puede tener diversas causas.
En la ejecución de un comando de escritura, de lectura o de Scan, se activa este flag en caso de que el controlador no pueda hallar el sector indicado.
En la ejecución del comando 'leer sector ID' se activa el flag ND si el controlador no puede efectuar una lectura carente de errores de un campo de ID. También en este caso la causa de error es un error en la suma de comprobación.
La tercera causa posible aparece con el comando 'leer pista' si el sector de inicio no fue hallado en la pista.

- Bit 1 NW:** Not Writable (Escritura no permitida)
Si en la ejecución de los comandos 'escribir sector', 'escribir sector borrado' o 'formatear pista' se detecta que el disco está provisto de protección de escritura, se activa este flag.
- Bit 0 MA :** Missing Adress Mark (Perdida Data Adress Mark)
Este flag se activa cuando el FDC no halla en la lectura de datos la ID de sector en el término de una revolución completa del disco. La falta de la Data Adress Mark o una Data Adress Mark borrada también se indican mediante la activación de este bit. Adicional y simultáneamente se activa el flag de MD en el registro de estado 2.

El registro de estado 2

Similarmetn a lo que ocurría en el ST1, en el ST2 se dan informaciones acerca del éxito o fracaso de un comando.

- Bit 7** no utilizado, siempre cero
- Bit 6 CM:** Control Mark
Si el FDC halla una Data Adress Mark borrada al efectuar una lectura de datos o en un comando de scan, activa este bit.
- Bit 5 DD:** Data Error in Data Field (Error en campo Data)
Similarmetn al flag de DE (bit 5 del ST1), este bit se activa ante errores de CRC. De hecho, este bit sólo se activa ante errores en campos de datos.
- Bit 4 WC:** Wrong Cylinder (pista erronea)
Al formatear una pista han de indicarse, para cada sector los números de sector, de pista y

de cabezal y la longitud de sector. Estos datos son almacenados en la ID de sector, y leídos dentro de un comando de lectura. Si el FDC detecta alguna diferencia entre el número de pista leído y el dado activa el WC-flag.

- Bit 3 SH:** Scan Equal Hit
Si se inicia un comando de Scan que compruebe la igualdad de la información de sector con los datos suministrados por el procesador, se activa este bit si los datos son iguales.
- Bit 2 SN:** Scan not Satisfied
Si, en un comando de Scan cualquiera, el FDC no halla ningún sector que corresponda a las indicaciones, se activa el bit de SN.
- Bit 1 BC:** Bad Cylinder
Su significado es similar al del flag de WC. Se activa cuando el número de pista leído del ID es &FF y no coincide con el de la orden.
- Bit 0 MD:** Missing Adress Mark in Data Field
Este flag se activa cuando el FDC no halla en la lectura de datos la Data Adress Mark o una Data Adress Mark borrada.

El registro de estado 3, el estado de la unidad

El contenido de este registro sólo puede ser transferido al procesador mediante la orden 'determinar estado de unidad'. Los bits de este registro reflejan el estado de la unidad elegida en el comando.

Bit 7 FT: Fault (Fallo)

El flag refleja el estado de la señal Fault existente en algunas unidades. Si la unidad dispone de tal conexión y el bit está activado, es que apareció un error en dicha unidad.

Bit 6 WP: Write Protected (Protección de escritura)

En este flag se puede ver si el disco insertado está protegido frente a escritura. Un flag de WP activado significa que el disco no puede ser escrito.

Bit 5 RY: Ready

Este bit se utiliza para mostrar el estado de la línea Ready de la unidad. Un flag de RY activado indica el estado 'Drive Ready'.

Bit 4 TO: Track 0 (pista 0)

Si el cabezal de lectura/escritura de la unidad elegida se halla en este momento del comando en la pista cero, el flag de TO estará activado.

Bit 3 TS: Two Side (Simple o doble cara)

En unidades de doble cabezal, esta conexión va a masa. En unidades sencillas, esta señal suele estar, por contra, a nivel alto. El programa puede reconocer en el estado del bit de TS qué tipo de unidad está conectada.

- Bit 2 HD: Head Adress (Cabezal)
Este bit refleja el estado de la Head Select-señal del FDC (patilla 27).
- Bits 1,0 US: Unit Select (Unidad)
Los estados de estos dos bits son idénticos a los niveles en ambas líneas de US del FDC (patillas 28 y 29).

3.1.2.4 Inserción del FDC 765 en el CPC

El fabricante, lamentablemente, no aprovechó ni de lejos las posibilidades del FDC al máximo. Así podemos conectar únicamente dos de las cuatro unidades en un principio posibles. Tampoco resulta posible el funcionamiento de unidades de doble cabezal, por cuanto la señal HEAD-SELECT es efectivamente enviada, pero no utilizada. Peor aún es el caso de la señal HEAD LOAD, que no va conectada a parte alguna. Esta carencia puede ser perdonada, no sólo porque el servicio de unidades de 8" resulte poco interesante para el usuario 'medio' por los enormes tamaños de estas unidades, sino también porque otros detalles de circuitería resultan inviables en el controlador.

A pesar de estas limitaciones, el controlador está muy bien realizado para la utilización elegida, la del servicio libre de problemas de dos unidades de 3". Con un gasto de hardware mínimo se creó un controlador que ofrece unas excelentes prestaciones.

Por fortuna, dentro del ahorro de costes de todo fabricante, en este caso concreto no se ha limitado la potencia del aparato. Se le ha incorporado al FDC un componente como 'ayuda' de un modo que reconocerán los expertos en hardware. Nos referimos al separador de datos integrado SMC 9229 (de 20 patillas en el CPC 464) o bien el SMC 9216 (de 8 patillas en CPC 664 y CPC 6128). El FDC y el separador de datos generan todas las señales para el floppy-interface aparte de las necesarias para la activación de los motores de la unidad.

Si bien el servicio de DMA supone el método más sencillo y elegante de conectar el controlador del floppy, se decidió seguir otra opción, por motivos económicos. El FDC es scrutado. Ello significa que el procesador es sincronizado con la transferencia de datos mediante los registros principales de estado. Las interrupciones generadas por el controlador no se utilizan. De hecho, la conexión de interrupciones del FDC no está conectada.

Respecto a direcciones, el FDC se halla en las direcciones de port &FB7E y &FB7F. En la primera dirección se halla el registro principal de estado, la segunda corresponde al registro de datos.

El Controller Board ocupa una tercera dirección. En el port &FA7E se halla un flip-flop a través del que se gobiernan los motores de la unidad. Si escribimos un 1 (OUT &FA7E,1 en BASIC), son encendidos los motores de todas las unidades conectadas; si escribimos un 0, son de nuevo apagados.

Resulta interesante la disposición en el circuito de la conexión de Terminal Count y de la patilla de Reset. Ambas conexiones están unidas y obtienen simultáneamente un impulso positivo en caso de Reset. En el controlador no está prevista la generación de un impulso de TC independiente. Ello plantea naturalmente la pregunta de cómo se finaliza un acceso a lectura de disco. (Nota: las explicaciones que siguen no son sólo válidas para la lectura de datos, sino que son extensibles a los comandos de escritura y de Scan.)

Lo usual es colocar un impulso de TC tras la lectura del sector deseado. Sin este impulso se ejecuta una E/S multisector, es decir, el controlador lee datos hasta que alcance el último sector de la pista. Pero dado que el último sector de la pista ha de ser programado dentro de los nueve bytes del comando de lectura, los fabricantes se sirvieron de un truco. Simplemente igualan el número del último sector al del sector a leer. Tras haber sido leídos todos los datos del sector, el FDC finaliza automáticamente la lectura y comienza la fase de resultados.

De este procedimiento se desprende una cuestión ante la que hay que estar precavidos a la hora de la programación de las rutinas del controlador. Si se lee un sector mediante este procedimiento, el registro de estado 0 replica al sistema operativo mediante un mensaje de error que reza: 'comando iniciado pero irregularmente finalizado', el bit 6 del ST0 está activado. El motivo concreto del error se halla en el ST1, donde el bit 7 está activado. Hablando claro: alcanzado el fin de la pista, intento de acceso a un sector tras el fin de la pista. Tal error ha de ser ignorado por el sistema operativo, pero sin despreciar otros eventuales errores.

La estructura de circuitos existente no permite el servicio de unidades de 8 pulgadas. Pero el hecho es que las rutinas en el AMSDOS tampoco están concebidas para el servicio de unidades de 8". Bajo AMSDOS no resulta posible, o es en todo caso muy difícil conectar aparatos de otras características como por ejemplo 80 pistas o doble cabezal. En lo fundamental, la conexión de tales unidades resulta posible; la señal correspondiente a elección de cabezal para unidades de doble cabezal es incluso enviada al floppy-interface. Pero en cualquier caso, partes esenciales del DOS han de ser reescritas, dado que el AMSDOS no soporta la HS señal.

3.2 La estructura del disco

3.2.1 Los tres formatos de disco

Como es sabido, el CPC distingue tres formatos de disco distintos, el formato CPC estándar, el formato de datos y el formato IBM. Puede elegirse el formato utilizado en el formateo del disco indicando S o V (por System y Vendor), D (por Datos) o bien I (por IBM). Vamos a ocuparnos algo más detalladamente de cómo están estructurados los formatos y en qué se diferencian.

En cada acceso a un disco, el AMSDOS determina por sí mismo el formato. De hecho ello sólo sucede cuando no hay ningún fichero abierto en el disco. Pero podemos prescindir de esta

limitaci n ya que, como es sabido, no deberian extraerse de la unidad discos con ficheros abiertos. El criterio de distinci n, los distintos n meros de sector en cada formato, ya fue comentado.

Otra posibilidad para soslayar este log-in autom tico, como tambi n se llama a la determinaci n de los par metros de disco, consiste en fijar ciertas posiciones de memoria de la RAM. Hallar  estas posiciones de memoria en el listado de la RAM del floppy en un capitulo posterior.

Los tres formatos tienen algunas caracteristicas comunes. En primer lugar, el n mero de pistas en el disco. Siempre es de 40, con las pistas individuales numeradas de 0 a 39. La longitud de sector tambi n es la misma en los tres formatos, con 512 bytes/sector. Aparte de ello, con cualquier formato caben como m ximo 64 ficheros por disco.

Pero aqu  se acaban las coincidencias.

3.2.1.1 El formato CPC est ndar o del sistema

Este formato es, probablemente, el m s usual y utilizado del CPC. Con este formato, una pista contiene 9 sectores con los n meros de sector &41 a &49. Aparte de ello, las dos primeras pistas del disco est n reservadas para el CP/M. As  que si desea trabajar en CP/M deberia utilizar este formato ya que s lo con el resulta posible un Start (Inicializaci n) o Warm Boot (Control C, arranque en caliente) del CP/M.

Las pistas reservadas est n dispuestas como sigue:

- Pista 0, sector &41: sector de arranque (Boot)
- Pista 0, sector &42: sector de configuraci n
- Pista 0, sectores &43 a &47: no utilizados.
- Pista 0, sectores &48 y &49 y
- Pista 1, sectores &41 a &49: CCP y BDOS

Si bien hemos hablado hasta ahora de tres formatos, lo cual es bàsicamente correcto, el programa 'FORMAT.COM' reconoce la entrada de un cuarto paràmetro, la 'V', además de S, D e I. Con ella se formatea el disco como con S, sólo que no se escribe nada en las pistas del sistema. Esta opción está pensada para la venta de software en CP/M, que ha de ser vendido sin CP/M por motivos de copyright. El usuario ha de escribir por sí mismo las pistas del sistema tras la compra. En ello puede ayudar el programa SYSGEN.COM.

3.2.1.2 El formato de datos

También en este formato hay 9 sectores formateados en cada una de las 40 pistas. Sólo que los números de sector se denominan de &C1 a &C9. En este formato no hay pistas reservadas para el sistema, de modo que quedan 9216 bytes más à su disposición.

3.2.1.3 El formato IBM

Este formato es idèntico al del PC de IBM bajo CP/M 86. En caso de que disponga de otro aparato además del CPC, podrá leer y escribir los discos del IBM también en su CPC. Ello supone naturalmente que ambos ordenadores trabajan con la misma medida física de floppy.

El espacio de memoria resulta ser algo menor en el formato IBM que en los dos anteriores ya que solamente se formatean 8 sectores en cada pista. Este reducido número de sectores queda en cierta medida compensado por el hecho de que únicamente se reserva una pista, la primera, como pista del sistema.

3.2.2 La estructura del directorio

En cualquiera de los tres formatos pueden almacenarse como máximo 64 ficheros en un disco. En este apartado trataremos con mayor profundidad acerca de dónde y de qué modo se almacenan en el disco los nombres de fichero.

La estructura fundamental del directorio viene impuesta por el CP/M. Dado que los discos han de ser leídos y escritos tanto bajo CP/M como bajo AMSDOS, este último hubo de ser ajustado a la estructura de directorio del CP/M. Así que, si bien hablaremos de AMSDOS en las siguientes explicaciones, debido a la compatibilidad de ambos sistemas operativos, dichas explicaciones serán también válidas para CP/M sin necesidad de que vayamos insistiendo en ello.

La elaboración automática de un directorio es una de las tareas esenciales de cualquier sistema operativo de discos. Sólo gracias a un directorio de este tipo se da la posibilidad de hallar tan rápidamente datos en el disco. Pero si únicamente se anotan los nombres de los ficheros, no se obtiene el rápido acceso deseado. En tal caso debería usted ocuparse de la correcta gestión de los sectores individuales en el disco. Se hará una idea de la complejidad de una tarea tal cuando, por cualquier motivo, el directorio de su disco preferido se haya vuelto completamente ilegible, y deba usted salvar los datos a mano.

Así que en el directorio debería anotarse, además del nombre del fichero, la posición física de los datos en el disco. Y son precisamente estos dos datos los que almacena el AMSDOS en una entrada de directorio de un fichero. Antes de ocuparnos más profundamente del almacenamiento, es preciso aclarar algunos conceptos que se utilizarán frecuentemente en lo sucesivo.

En primer lugar tenemos el concepto de SECTOR. Un sector es la zona del disco dispuesta para los datos al formatear. En el AMSDOS los sectores tienen siempre una longitud de 512

bytes, otros sistemas operativos utilizan sectores de 128, 256 ó incluso 1024 bytes.

Si se han de leer datos del disco, siempre ha de leerse un sector completo. No resulta posible leer precisamente unos cuantos determinados bytes de un disco. El sector es, por tanto, la mínima zona de memoria a la que podemos referirnos.

Un RECORD es un bloque menor de datos, de exactamente 128 bytes. Cada sector (del AMSDOS) contiene por ello exactamente 4 records. ¿Y para qué esta división? Pues la justificación se halla en la historia de la evolución del CP/M. El CP/M fue originalmente desarrollado para computadores con unidades de disco de 8". Al principio, los sectores todavía eran, en estas unidades, de 128 bytes. Fue después cuando se desarrollaron formatos en los que los sectores empezaron a ser mayores de 128 bytes. Para conservar la compatibilidad con el formato anteriormente utilizado, el BIOS dividió simplemente mediante técnicas de programación el sector mayor en unidades menores, de 128 bytes. Con ello se salvaguardó la compatibilidad. Lógicamente, el AMSDOS también trabaja a nivel de records. Ello significa que, de hecho, ni el AMSDOS ni el CP/M saben de sectores.

Un tercer concepto que merece aclaración es el de BLOQUE. También este concepto proviene de los orígenes del CP/M. Si se desea almacenar en disco ficheros de unos 10 K, ha de anotarse una considerable cantidad de sectores ocupados por el fichero. Pero podemos reducir drásticamente dicho número reuniendo varios records en bloques y anotando los números de bloque. La longitud de los bloques es de libre definición en CP/M, siendo usuales valores de 1 K (también en AMSDOS) o de 2 K. Para superar la cantidad de cálculos necesarios para ello, se numeran simplemente todos los records libres —es decir, no ocupados por las pistas de sistema— relativamente comenzando por las pistas inferiores. En la práctica, un disco en formato S toma el siguiente aspecto: El sector &41 de la pista 2 contiene los records 0

hasta 3 del disco. En el sector &42 se hallan los records 4 hasta 7, y así sucesivamente. Dado que, como se ha dicho, un bloque en AMSDOS tiene una longitud de 1 K, contiene 8 records. De ahí que el bloque 0 haya de ser buscado en los sectores &41 y &42 de la pista 2, mientras que el bloque 1 ocupa los sectores &43 y &44, y, por ejemplo, el bloque 4, los sectores &49 de la pista 2 y &41 de la pista 3.

De acuerdo, estos cálculos son un tanto enrevesados, pero ineludibles bajo CP/M y AMSDOS. Volvamos al directorio.

¿No se ha preguntado nunca por qué cualquier programa le ocupa un mínimo de 1 K en el disco, incluso si es de un byte, lo que no llega a ocupar un sector, ni siquiera un record? Acabamos de conocer la causa. Son los números de bloque ocupados por el fichero, anotados en el directorio.

Observemos atentamente una entrada de directorio de ese estilo. Por cada entrada se precisan 32 bytes. Comencemos con los 16 primeros bytes.

00 46 4F 52 4D 41 54 20 20 43 4F 4D 00 00 00 15.FORMAT COM..

Se trata, en esta estrada DIR, de la anotación correspondiente al programa FORMAT.COM. Más no se puede decir sin un conocimiento detallado del AMSDOS. El punto entre el nombre de fichero y la extensión aparentemente no es almacenado ya que el nombre de fichero es relleno mediante dos espacios (valor ASCII 32 ó 20 hex). Pero ¿qué indica el valor de 0 ante el nombre de fichero? Acuérdate del número de user, que podía ser incorporado a cada fichero. Este número se anota en el primer byte de la entrada y regula el acceso al fichero. Este valor tiene otro significado especial aparte del anterior, que ya veremos más adelante.

Siguen en el nombre de fichero 3 bytes nulos y 1 byte con el valor de &15. No resulta fácil obtener el significado de los mismos. Solamente son importantes los bytes 12 y 15, pero todavía no revelaremos su significado. Observemos antes los 16 bytes restantes de la entrada.

55 56 57 00 00 00 00 00 00 00 00 00 00 00 00 00 UVW.....

Tampoco aquí resulta fácil entender el significado de estos bytes. En cualquier caso, el intentar una orden CAT con el disco master de CP/M insertado nos ayudaría algo a solucionar el problema. En el catálogo aparece el fichero FORMAT.COM, con una longitud de 3 K. Dado que 1 K corresponde a un bloque y tres de los 16 bytes contienen un valor distinto de 0, resulta razonable sospechar que se trate de los números correspondientes a los de bloque. Y así es. De hecho, la disposición de los bloques de un fichero se anota en los bytes 16 a 31 de una entrada de directorio.

Ello da pie a un montón de nuevas preguntas; la principal de ellas debería ser: ¿Qué sucede cuando un fichero es mayor de 16 K? La respuesta es bien sencilla; el AMSDOS proporciona a los ficheros de tal magnitud simplemente una entrada de fichero suplementaria. Una entrada de ampliación, Extent; extensión se diferencia en escasos aspectos del primer asiento. En esencia se anota la disposición de bloques de continuación en los bytes 16 a 31. El número de usuario y el nombre de fichero son idénticos en ambas entradas.

La siguiente pregunta que se plantea reza: ¿En qué reconoce el AMSDOS que ha de seguir una extensión? Para ello hemos de conocer el significado del byte 15 de una entrada. Se precisa un simple ejemplo de cálculo.

Nuestro ejemplo FORMAT.COM consta de tres bloques. Como ya hemos dicho, un bloque contiene 8 records. Así que el fichero FORMAT.COM puede constar como máximo de 24 records (hex 18). El número de records de FORMAT.COM es de 15 hexadecimal.

Si un fichero es tan grande que precisa más de una entrada, se calcula el valor en el byte 15 de la entrada mediante $16 \text{ (bloques)} * 8 \text{ (records)} = 128 \text{ ó } &80$. Cuando en esta posición se halte dicho valor, el AMSDOS supondrá automáticamente de ello que todavía sigue una extensión.

Para no complicarnos con ficheros especialmente grandes con múltiples extensiones, se numeran dichas extensiones en orden creciente en el byte 12, es decir, inmediatamente después del nombre de fichero. Con ello se concreta el orden de lectura de las extensiones. Mediante esta organización, un fichero puede llegar a ser tan grande que no haya espacio para nuevas entradas en el directorio, o llenar el disco.

Pero volvamos a hablar del primer byte del asiento. Como ya se comentó, en él se anota el número de usuario del fichero. Si prueba el programa para la lectura de sectores cualesquiera o el monitor de disco inspeccionando los sectores del directorio de algunos de sus discos, probablemente hallará en algunos ficheros el valor de &E5 en lugar de un número válido de usuario (0 a 15). Pero mediante la orden CAT no aparecen en el directorio los nombres de fichero correspondientes. No se sorprenda, ya que es más que altamente probable que los haya borrado usted mismo con ayuda del comando ERA.

Y es que el AMSDOS es muy precavido al borrar ficheros. Unicamente se busca el nombre de fichero dado, con todas sus eventualmente existentes extensiones y se cambia el valor del número de usuario a &E5. A partir de ahora, dicho fichero no existe para el AMSDOS, aun cuando todos los datos sigan estando en el disco. Este cauteloso proceder del AMSDOS resulta muy agradable tras haber borrado inadvertidamente ficheros importantes. En un caso tal, lea simplemente los sectores del directorio mediante un monitor de disco y modifique los números de usuario del fichero borrado a un valor razonable. Tras reescribirlo, resucitan los ficheros. El valor &E5 no fue tomado casualmente como marca de borrado. Tras el formateo, todos los sectores están escritos con el valor &E5, también por tanto los sectores DIR. Si ahora se efectúa un acceso al directorio, el AMSDOS halla para cada posible entrada el indicativo sin precisar procesos especiales.

Ello nos lleva al último punto importante del directorio, la posición en el disco. Como ya mencionamos, en el directorio pueden anotarse hasta 64 ficheros. Con un gasto de 32 bytes

por entrada resultan necesarios 2048 bytes. Ello supone 2 bloques, o bien 4 sectores o también 16 records. En esencia se utilizan en cada uno de los tres formatos los dos primeros bloques de la primera pista libre. Así que en el formato S el directorio estará en los sectores &41 hasta &45 de la pista 2, en el formato D en los sectores &C1 hasta &C5 de la pista 0 y en el formato I en los sectores &01 hasta &05 de la pista 1.

Y para finalizar el capítulo sobre el directorio, dos consejos.

Si el bit 7 del primer carácter de una extensión, es decir del noveno byte de la entrada, se activa, el fichero contiene el atributo READ-ONLY (sólo lectura). Los ficheros así marcados no pueden ser ni borrados ni renombrados bajo AMSDOS y CP/M. Para activar el bit existen (al menos) dos caminos distintos. La primera alternativa la ofrece el comando transeunte CP/M STAT. La segunda alternativa la ofrece el monitor de disco. Tras localizar la entrada a proteger en los sectores de directorio sume al valor ASCII del primer carácter el valor 128 ó &80, modifique el byte correspondientemente al resultado y escriba el sector simplemente de nuevo.

Con nuestro segundo consejo, puede usted proteger un fichero de miradas indiscretas. Para ello ha de fijar, similarmente al caso del atributo de READ-ONLY, el bit 7 del segundo carácter de la extensión. También esta tarea puede realizarse bajo CP/M mediante STAT o bien con nuestro monitor de disco. Pero no olvide el nombre de los ficheros protegidos de esta forma. No lo conseguiría rescatar ni con CAT ni con DIR.

3.2.3 La estructura de los ficheros

Tras conseguir el capítulo anterior una visión global sobre la información almacenada en el directorio, veamos de qué forma están realmente almacenados los datos en el disco.

En primer lugar surge la pregunta de dónde se recoge en la lectura de un fichero información, como el tipo de fichero, la longitud del fichero o la dirección de inicio del programa cargado. Estas informaciones de cabecera no están contenidas en la entrada de directorio. Así que han de estar almacenadas junto con los datos.

Pero ello plantea el problema de saber en qué forma han de ser almacenados estos datos adicionales sin que se produzcan confusiones entre la cabecera y los datos. Llevaremos a cabo algunos intentos que nos darán la clave acerca del modo de almacenamiento.

Introduzca la línea

```
SAVE "X1.BIN",B,&1000,1024
```

en su CPC y pulse la tecla ENTER. No nos interesa, por el momento, el contenido del fichero, por lo que puede elegir cualquier dirección como valor de inicio. Lo que sí resulta importante es la longitud de la zona de memoria. Como ya vimos, los ficheros se almacenan bajo AMSDOS siempre en bloques de 1 K. Dado que hemos elegido una longitud de exactamente 1 K, la longitud del fichero que muestra el directorio debería ser de 1 K. .Que más quisieras! El fichero tiene 2 K.

Vuelva a introducir la linea, pero rebajando el valor de la longitud de fichero en 100, dejándola en 924. Para sorpresa general, el fichero, aparentemente, aún no se ha reducido, pues sigue mostrando 2 K.

Hagamos un tercer intento. Introduzca como longitud la expresión 1024-128. Y... abracadabra .conseguido! Nuestro fichero X1.BIN ocupa solamente 1 K en el disco. Pero cualquier incremento en la longitud, aun de un único byte, lleva de nuevo a los 2 K.

Así que se almacena un record adicional, o, dicho de otro modo, 128 bytes. En este record se hallan todas las informaciones de cabecera. El record de cabecera es el primer record de casi cualquier fichero. Y ¿por qué así? Pruebe el siguiente programa:

```
10 OPENOUT "X1.DAT"
20 FOR i=1 TO 1024
30 PRINT#9,"x";
40 NEXT i
50 CLOSEOUT
```

De lo que hasta ahora conocemos se podría sospechar que el fichero X1.DAT ha de aparecer en el directorio con 2 K. Pero no es así. De hecho, el fichero sólo ocupa 1 K, exactamente la longitud dada por el número de caracteres escritos en él.

De ello se desprende la siguiente regla:

Todos los ficheros ASCII puros, los almacenados con el indicativo &16, carecen de record de cabecera. (Nota: El sistema operativo sólo valora el nibble bajo, es decir, &16 y &36 son igualmente tomados como ficheros ASCII puros.)

En todos los demás ficheros, p.ej. programas BASIC o en lenguaje máquina, el primer record es el record de cabecera, que es copiado en la carga en la zona de cabecera.

Hasta aquí, perfecto. Pero, ¿cómo puede determinar un programa si el fichero es un fichero ASCII puro? A fin de cuentas, en el directorio no hay información alguna acerca del tipo de fichero. Este problema le habrá costado al programador del AMSDOS más de un quebradero de cabeza. Finalmente se tomó un camino que parece aceptable.

En la apertura de un fichero de entrada se efectúa una suma de comprobación de 16 bits sobre los 66 primeros bytes del primer record. El resultado se compara con el contenido de los bytes 67 y 68 del record. Si los valores coinciden, este record será, con relativa seguridad, un record de cabecera. En caso de que se cumpla casualmente esta condición de comprobación en un fichero ASCII, se pierde el primer record del fichero. Pero la probabilidad de que esto ocurra es infima, por lo cual no se comprueba.

3.2.4 El registro físico de los datos en el disco

¿Es usted del tipo de personas a las que les gusta conocer el fondo de las cuestiones? Pues este capítulo le irá al pelo. Vamos a mostrarle de qué modo se hallan realmente los datos en el disco. Quede claro que en este capítulo entendemos por dato no sólo el contenido de un fichero, sino todo aquello que es alojado de algún modo en una pista. Aclararemos también conceptos como los de Sector-ID, Gap o Adress Mark.

3.2.4.1 MF, MFM, bits y magnetismo

Como ya sabe, la superficie del disco está dividida en 40 coronas circulares concéntricas o pistas. El cabezal de escritura/lectura de la unidad de disco puede ser emplazado con precisión sobre cada una de estas pistas con ayuda de un motor paso a paso. De modo similar a como ocurre en una grabadora, los datos se almacenan sobre el disco en forma de bits. Pero el almacenamiento de datos en forma de bits y la utilización común de soportes magnéticos son todos los rasgos comunes entre ambos sistemas.

En el floppy se magnetiza totalmente la capa magnética del disco. En el cassette teníamos la mayor sobreexcitación posible con un factor de distorsión no lineal del 100 %. Y sin embargo, este factor 'ideal' no molesta en absoluto en el floppy. Al contrario, cuanto mejor funcione la magnetiza-

ción en la escritura, más sencilla resultará la lectura de los datos.

Para entender el formato de registro empleado en el CPC, hemos de realizar una breve incursión en la Física. El cabezal de escritura/lectura de una unidad de disco es, en esencia, una bobina. Las bobinas son de los elementos más interesantes de la Electrónica, si bien de los más olvidados. Una de las más importantes características de las bobinas es el hecho de que un campo magnético variable induce en dicha bobina una tensión. También es importante indicar que el campo magnético varía constantemente para generar una tensión constante en la bobina. Es el mismo principio que hallamos, por ejemplo, en la dinamo de la bicicleta.

También resulta interesante de esta propiedad su reciprocidad. Una tensión alterna variable aplicada a una bobina induce en la misma un campo magnético variable con el mismo ritmo.

Ambas propiedades de la bobina son utilizadas en el cabezal de escritura/lectura de la unidad de disco. Para escribir una información en el disco, se aplica una tensión alterna al cabezal, la cual crea un campo magnético. Este campo magnetiza el recubrimiento del disco, produciendo la grabación. En el caso inverso, mediante el campo magnético variable del disco se crea una tensión, que es ampliada y preparada por una electrónica adecuada.

Por lo hasta ahora expuesto, podríamos caer en la idea de representar un bit bajo ó 0 por la ausencia de magnetización y un bit alto ó 1 por la presencia del campo magnético. Con ello se obtendría en la lectura directamente de vuelta la señal digital. Pero no es tan sencillo. La electrónica necesaria para ello tendría que trabajar con una extrema precisión, dado que, con múltiples bits altos y bajos, uno detrás del otro, tendrían que medirse y valorarse exactamente los tiempos respectivos para bajo y alto.

Resulta más sencillo, si junto a los datos se registra una frecuencia (de reloj) de referencia concreta. Mediante esta frecuencia de referencia resulta muy fácil determinar una celda de un bit. Pero el registro de campos magnéticos variando rápidamente en el caso de bits altos tampoco se ha de utilizar de este modo. En lugar de ello, se genera un impulso exactamente entre dos impulsos de referencia; en caso de un cero, simplemente no hay nada ahí. El aspecto de una cadena de impulsos tal es el que refleja el diagrama de la figura 32.

Mediante esta cadena de impulsos se dirige un flip-flop que con cada impulso de entrada invierte el estado de su salida. La tensión de salida resultante es la del diagrama de la figura 32.

Esta señal de salida resultante es especialmente adecuada para el control del cabezal de escritura/lectura. Si la tensión de salida del flip-flop es alta, a través de la bobina fluiría una corriente en un determinado sentido, y la capa magnética del disco se magnetizará en un cierto sentido. Pero si la tensión de salida del flip-flop varía, se invierte el sentido de la corriente en la bobina, lo que también provoca la inversión del sentido de magnetización en el disco.

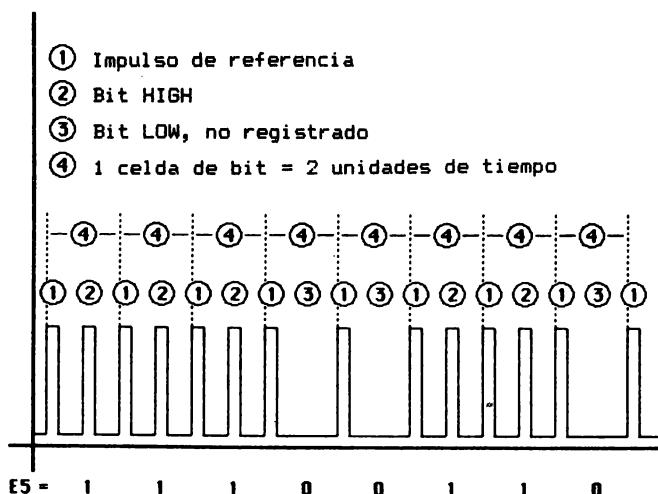


Figura 32.

Gráfica de la señal en el cabezal R/W en escritura
(Simple densidad)

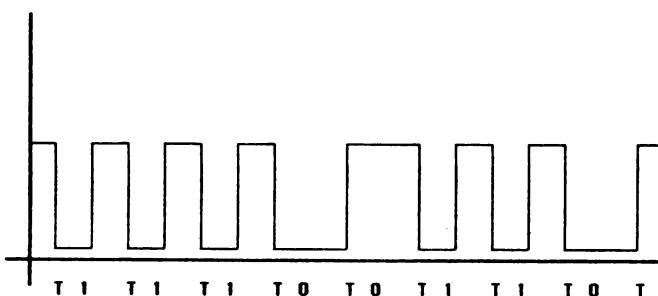


Figura 33.

Si los datos se registran así, es decir por diferencias alto/bajo, en el disco tenemos finalmente muchos microimanes, magnetizados en un sentido u otro según registro. La figura 34 representa gráficamente este encadenamiento de pequeños imanes.

Pero, ¿qué sucede en la lectura de los datos? Hemos dicho que el disco es completamente magnetizado. Mediante la rotación del disco la multitud de microimanes pasa por el cabezal de lectura. Con cada variación del campo magnético aparece en la bobina un impulso, que puede ser amplificado y preparado. Entretanto, el campo magnético no varía prácticamente nada, tampoco se genera tensión alguna. La figura 35 representa la señal de salida amplificada. La similitud con el primer diagrama no es casual. Hemos leído nuestra información de nuevo íntegramente del disco.

La electrónica de valoración ha de utilizar con este procedimiento únicamente dos tiempos. El primero y más largo es el lapso entre dos impulsos de referencia, que aparece siempre ante bits bajos, el segundo especifica el periodo entre el impulso de referencia y el de datos en bits altos. Dado que los tiempos se conservan con relativa precisión, pueden separarse los impulsos de datos de los de referencia con la ayuda de sencillos circuitos Mono-Flop, dado que siempre hay una referencia disponible para la sincronización.

Algo más sobre los tiempos que intervienen: un impulso de referencia dura exactamente 500 nanosegundos (media millonésima de segundo, lo que con relojes de pulsera no es en absoluto mesurable), el intervalo entre dos impulsos de referencia es de 8 microsegundos. Así que un byte precisa $8 * 8 = 64$ microsegundos para escribir o leer con el procedimiento descrito. Con la velocidad nominal de una vuelta del disco cada 200 milisegundos, caben en el disco 40 (pistas) * 200 (milisegundos)/64 (microsegundos/byte) = 125000 bytes.

¿Qué en su disco caben de 40000 a 50000 bytes más? Bueno, de hecho, lo que hasta ahora le hemos descrito es el formato de

simple densidad o MF. Basandonos en éste pasaremos a continuación a describirles el formato de doble densidad o MFM, el utilizado en el CPC.

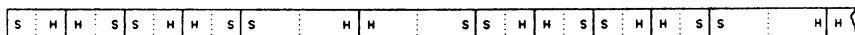


Figura 34.

SEÑAL DE LECTURA AMPLIFICADA

- ① Impulso de referencia
 - ② Bit HIGH
 - ③ Bit LOW, sin variación en el cambio de flujo

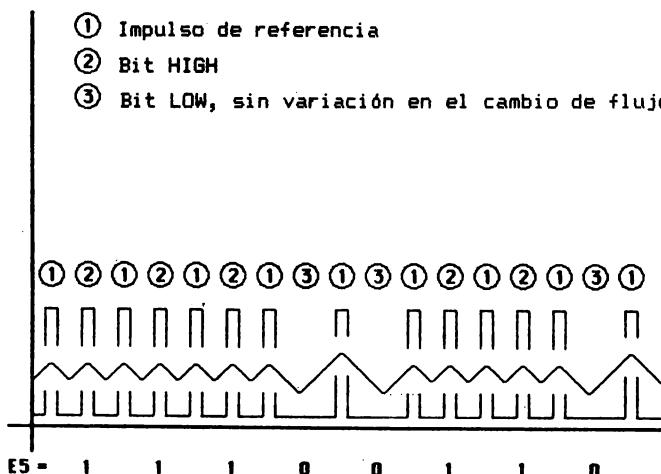


Figura 35.

Algunas mentes despiertas no se conformaron con el procedimiento anterior. Se siguió investigando hasta desarrollar un procedimiento, viable en todo caso, para casi doblar la capacidad de almacenamiento. Se partió de la idea de que debía resultar posible renunciar de vez en cuando a los impulsos de referencia, aunque no sea posible hacerlo siempre. El registro de bits altos no plantea dificultades especiales dado que cada bit 1 a registrar provoca un cambio de flujo en el disco. Pero lamentablemente, también tenemos los bits 0, que han de ser registrados. Si lo que tenemos es una secuencia de bits nulos, no hay sincronización.

Para evitarlo, se registran simplemente bits de referencia ante varios ceros consecutivos. De nuevo se distinguen las diferencias entre los bits de referencia y de datos mediante tiempos, que de hecho son tres. Los hallará en las representaciones gráficas 36 y 37, donde el primer diagrama corresponde a la información de entrada y la segunda al registro en el disco.

Un intervalo elemental de tiempo es el tiempo entre dos unos consecutivos, un intervalo de tiempo de longitud doble representa la secuencia de bits 1-0-1. Si siguen varios ceros consecutivos, se precisa del tercer lapso posible. Este tiempo, de una unidad y media de duración, denota el paso de impulso de referencia a impulso de datos o viceversa.

Por desgracia, en los gráficos no se aprecia demasiado bien la ventaja inmediata del procedimiento MFM, debido a la escala elegida. Pero tenga en cuenta que la división del tiempo no es la misma en ambas representaciones. Si se considera que mediante este procedimiento se trabaja el registro físico con el mismo número posible de cambios de flujo, es decir variaciones del campo magnético, resulta una duplicación efectiva de la capacidad del disco.

Naturalmente, este procedimiento exige una electrónica de valoración algo más precisa que en el formato sencillo MF. Pero en el CPC es el ya mencionado separador de datos el que realiza dicha valoración de un modo absolutamente fiable.

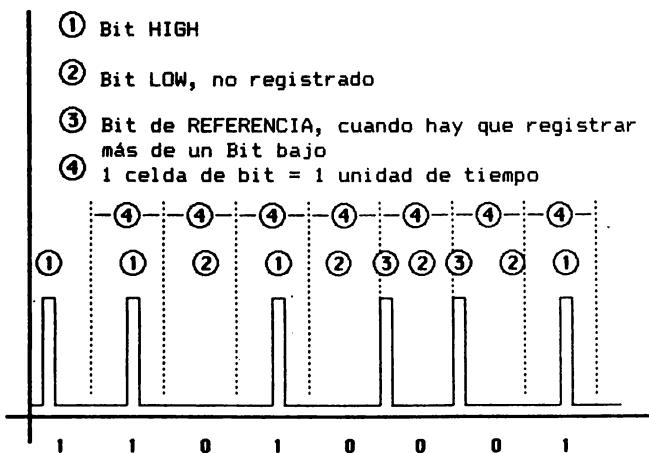


Figura 36

Gráfica de la señal en el cabezal R/W en escritura
(Doble densidad)

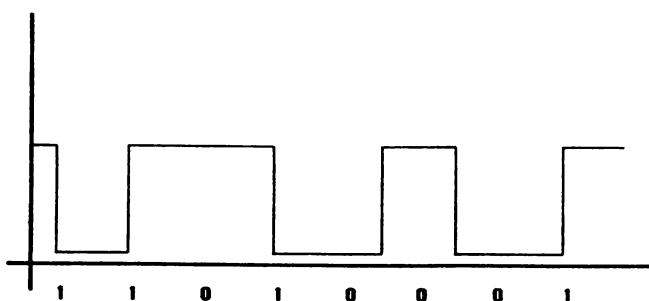


Figura 37

Pero todavía queda una pregunta en el aire. Dado que un disco formateado en formato MF tiene una capacidad de 125000 bytes, un disco en formato MFM debería poder almacenar 250000 bytes. Pero sólo se anuncian unos 182000 bytes como capacidad. ¿Dónde queda el RESTO?

3.2.4.2 GAPS, IDs y Adress-Marks

En el apartado anterior dividimos el disco en pistas individuales. Ello fue necesario para la utilización de un motor paso a paso para el control del cabezal. En este apartado dividiremos el disco como si de un pastel se tratara, sin cuchillo, en trozos individuales, los sectores.

Si no realizáramos esta división, no podríamos poner en una pista más que un fichero. Con ello se ocuparían siempre 6250 bytes, independientemente de la longitud que realmente precisara el fichero. Y en ficheros muy pequeños, no es éste precisamente el procedimiento más económico. Al trabajar con ficheros secuenciales tendría que prepararse en la memoria del ordenador un buffer de al menos 6250 bytes. Si se ha de leer de un fichero y escribir simultáneamente en otro, la memoria precisa se eleva a nada más y nada menos que a 12500 bytes. Demasiado incluso para un ordenador con 64 K.

Como se ve, la idea de la división en zonas menores es totalmente razonable. Pero para separar claramente estas zonas entre sí ha de realizarse un cierto gasto consistente en algo de espacio del disco, de modo que ya no disponemos de la capacidad de almacenamiento de datos original. El primer dato es la capacidad antes de formatear, la capacidad realmente disponible es la capacidad tras formateado.

Nos ocuparemos con más detalle del formateado y de la capacidad resultante. Hallará los procesos descritos

representados en la figura 38. Representa el contenido completo de una pista.

El reconocimiento del comienzo físico de una pista mediante la perforación de índice en el disco y un sensor óptico correspondientemente ajustado no supone ninguna dificultad especial para el FDC. El sensor óptico de índice genera un impulso cuando el rayo de luz llega al receptor a través de la perforación de índice. El fin del impulso es la señal para el FDC de comenzar inmediatamente con el formateo. Su primera acción consiste en escribir en el disco 80 bytes con el valor &4E. Esta zona se denomina GAP 4A.

La traducción literal de GAP es hueco. ¿Para qué sirve este hueco? La respuesta es bastante explícita. El hueco iguala tolerancias entre distintas unidades de disco. A pesar de la indiscutiblemente alta precisión de la unidad, se dan pequeñas diferencias en el ajuste de los sensores ópticos de índice entre distintos aparatos. El GAP 4A es de tal magnitud que ciertas diferencias de ajuste dentro de un margen concreto sean irrelevantes. Gracias a ello pueden leerse y escribirse los soportes de datos -es decir, los discos- en distintas unidades de disco.

Tras el GAP 4A se escribe una zona de Sync (sincronismo) de 12 bytes nulos. Como ya sabe, con varios bytes 0 se indica la frecuencia de reloj, tanto en el formato MF como en el MFM. La indicación SYNC significa que el FDC es sincronizado con el reloj de referencia.

Tras los 12 bytes de Sync, el FDC escribe tres bytes indicados como Index-Address-Mark. Estos tres bytes siguen un esquema muy especial y pueden ser reconocidos por el FDC mediante un hardware especial en el chip. La marca de dirección de índice se escribe sólo una vez al comienzo de la pista, es decir, inmediatamente detrás de la perforación de índice y supone en cierto modo una señal adicional para el comienzo de la pista. Tras esta marca se formatea un byte con el valor &FC y otro GAP, el GAP 1 con 50 bytes &4E. Este

GAP se precisa para dar tiempo suficiente al FDC para la elaboración del Index-AM en posterior lectura. Con ello está formateado el típico comienzo de pista. Ahora vienen zonas extras para cada sector.

Cada sector comienza con 12 bytes de Sync. Al igual que antes, sólo se escribe la información de reloj. Tras los bytes de Sync, sigue una marca de dirección, esta vez la ID-Adress-Mark. También la ID-AM es de 3 bytes y puede ser decodificada mediante hardware por el FDC. Tras la ID-AM viene un byte con el valor &FE.

Hasta aquí, todos los sectores del disco son idénticos. Pero para poder acceder posteriormente a las pistas y sectores a voluntad se hace preciso marcarlos también. Y es precisamente de esta tarea de la que se encargan los 4 bytes siguientes, llamados campo de ID. En el orden de la descripción se apuntan aquí el número de pista, la cara del disco (es decir 0 ó 1), el número de sector y la longitud de sector. Por supuesto que el último dato no puede ser directamente apuntado, ya que para sectores de 512 bytes se precisaría un campo de 2 bytes. La clave correspondiente corresponde a los datos dados al formatear. Los sectores de 512 bytes se marcan con un 2 en este byte.

Con ello se le ha dado a cada sector una identificación especial, la ID de sector, de cuya mano resulta posible un acceso posterior dirigido. Pero el diseñador del sistema no ha dejado ningún cabo suelto. Para estar protegidos frente a eventuales errores, mediante los tres bytes de la ID-AM, el &FE y de los 4 bytes del campo de ID, se realiza un suma de prueba de 2 bytes según un procedimiento especial. Este procedimiento se denomina Cyclic Redundancy Check o CRC y permite el reconocimiento de errores de lectura, siendo de una alta fiabilidad. Los dos bytes de CRC obtenidos se escriben inmediatamente tras el campo de ID.

Para darle tiempo al FDC en una posterior lectura de los datos para la comprobación de la ID y de los bytes de CRC, le sigue en primer lugar otro GAP, el GAP 2 con una longitud de 22 bytes. El valor de los datos (intrascendente) vuelve a

ser &4E. El GAP 2 tiene, además, otro significado. Todos los datos de un sector son escritos en una posterior escritura del mismo tras el GAP 2. El GAP 2 permite al FDC llevar a cabo la conmutación entre lectura y escritura, lo que siempre supone algo de tiempo.

¿Y adivina lo que sigue al GAP 2? Exacto, una zona de sync de 12 bytes nulos, que informa por fin del comienzo de la auténtica zona de datos. Pero todavía no se escriben en el disco los datos. Al sync le sigue siempre una zona de marca de direcciones. Este AM que se formatea ahora es la DATA-Address-Mark, de la misma estructura que la AM de ID y que puede ser decodificada del mismo modo por el FDC. Como distinción respecto a la AM de ID, en la AM de DATA sigue un byte con el valor &FB.

Y ahora sí. Por fin comienza la zona de datos, que recibe en el formateado el valor del byte de relleno. La longitud de esta zona es variable, pero fijada al formatear mediante los datos dados al FDC. Son posibles zonas de datos de entre 128 y 4096 bytes. Pero en el último caso, en una pista cabría un único sector, por lo que casi siempre se elegirán longitudes de sector menores. En el CPC dicha longitud de sector está fijada en 512 bytes.

También en la zona de datos se efectúa una suma de comprobación de 2 bytes según el procedimiento CRC. Para reconocer también errores en la DATA-ID, estos bytes se incluyen en la formación del CRC, de modo que con una longitud de sector de 512 bytes se comprueban en total 516 bytes.

El final de un sector conforma el GAP 3. La longitud de este GAP puede ser indicada en el formateo. En el CPC se escriben al formatear 82 bytes &4E. Este GAP tiene un significado muy especial. Si en un disco formateado se sobreescreiben posteriormente sectores con nuevos datos, resultaría totalmente improbable que el disco girase exactamente a la misma velocidad que tenía en el formateado. Pero si la

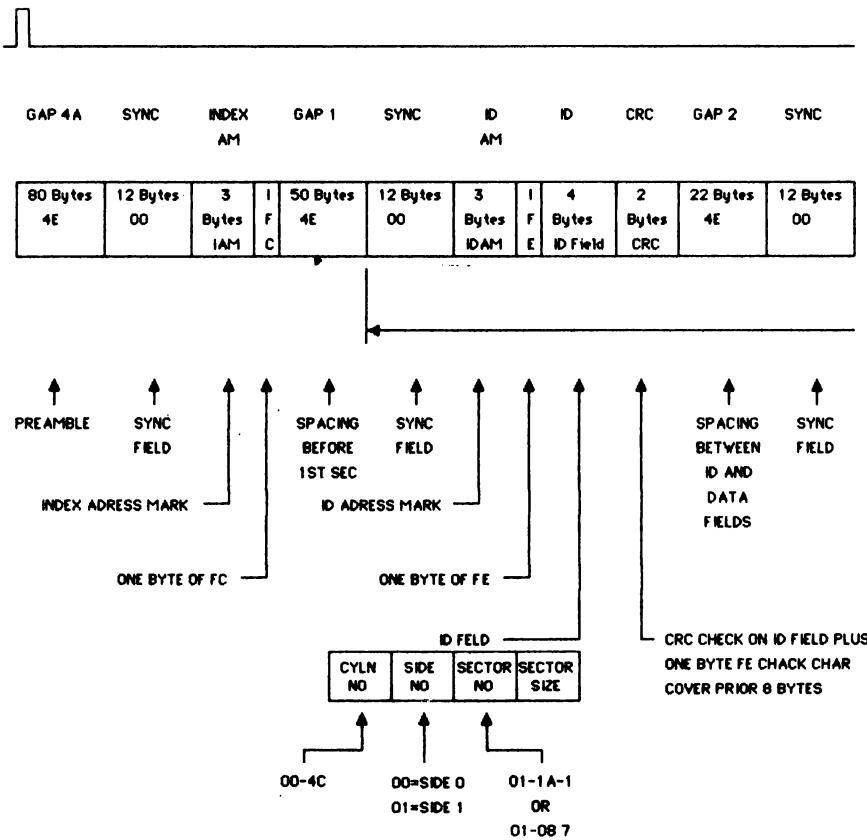
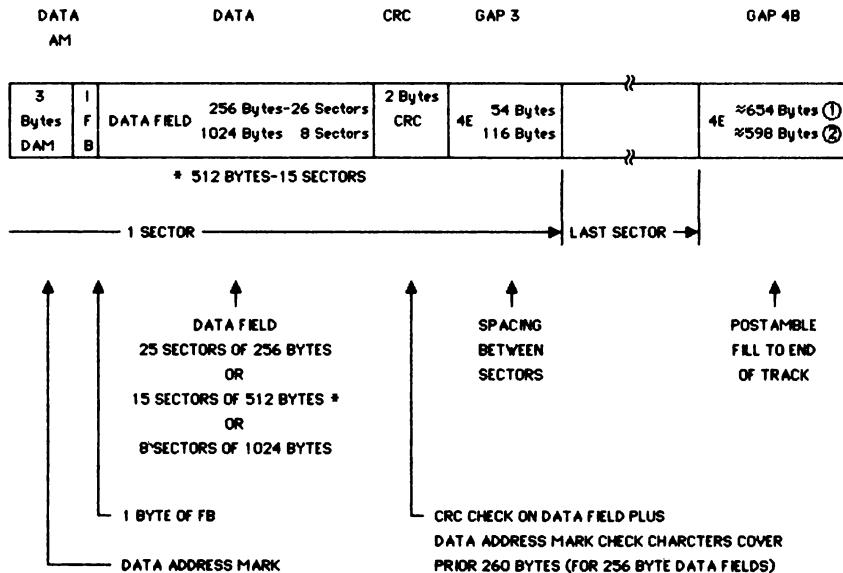


Figura 38

Técnica del floppy y del disco

INDEX
PULSE



- ① FOR 256 BYTES/SECTOR
- ② FOR 1024 BYTES/SECTOR
- * 512 BYTES -15 SECTORS
NOT AN IBM STANDARD
BUT OFTEN USED

velocidad es ligeramente mayor, el arco precisado por el sector se alargará ya que la velocidad de registro es constante. Sin el GAP 3, un tal registro de mayor longitud simplemente sobreescribiría un sector de ID inmediato, el sector siguiente dejaría de ser legible. Pero las diferencias de longitud generadas por las oscilaciones de la velocidad pueden ser interceptadas mediante el GAP 3 aunque también se genere el GAP 3 en la escritura de un sector. En cualquier caso se reduce drásticamente la longitud de un GAP generado en escritura de sector. De hecho se escriben tan sólo 42 bytes de GAP, de modo que queden suficientes para las diferencias que aparecen.

Con ello queda completamente formateado un sector, el FDC reclama ahora los valores para el siguiente sector y escribe también éste en el disco. Formateados en secuencia todos los sectores, se escriben bytes de GAP hasta la aparición del impulso de índice, con el cual finaliza el formateado de una pista.

Como se ve, el FDC tiene bastante que ver con la preparación de una pista para su uso. Otros FDC distintos al 765 nos pueden ayudar bastante menos. Así, por ejemplo, los frecuentemente incorporados controladores de la serie 197x precisan recibir casi cada byte del procesador, el programa de formateo se hace muy extenso y general. Pero gracias al gran apoyo del 765 el formateo con este FDC es casi un juego de niños.

Capítulo 4

La ROM y la RAM del AMSDOS

En primer lugar hemos de aclarar que no todo el mundo ha de ocuparse imprescindiblemente de las interioridades de un sistema operativo. Para muchos propietarios de ordenadores resulta totalmente suficiente con que los programas existentes funcionen sin errores y con un amplio margen de confianza. Tales propietarios pueden saltarse el presente capítulo sin temor alguno. Pero si las posibilidades que pone a nuestra disposición el AMSDOS llegaran a no bastarle para tareas concretas, debería acordarse de las páginas siguientes. Hallará aquí, en forma compacta, las llamadas de salto esenciales del sistema operativo del floppy y descripciones detalladas de cada una de las rutinas. Si algún comentario le resulta poco claro, puede desensamblar el sistema operativo con ayuda de un desensamblador. Junto con los comentarios, seguro que entonces podrá conseguir rápidamente la información deseada.

4.1 La RAM de sistema del AMSDOS

Como ya sabe, al encender el CPC y el floppy dispone usted de una memoria de 42249 bytes para programas BASIC. Pero sin el floppy, es decir, con el AMSDOS desactivado, son 43533 bytes los que ofrece el CPC. La diferencia de 1284 bytes no se debe exclusivamente al floppy. Este se reserva, al conectar, la mayor tajada, de 1024 bytes. Esta zona de 1024 bytes suele iniciarse en &A700. La limitación tiene un motivo especial en la gestión de ROMs externas a través del CPC. El CPC puede gestionar un máximo de 252 ROMs externas en la zona de memoria de &C000 hasta &FFFF. En ello se distinguen tres tipos distintos de ROM, las ROM de primer plano, las ROM de fondo y las ROM de extensión. El primer byte de una cierta ROM (&C000) es decisivo para el tipo de ROM.

Tras conectar, se comprueba en primer lugar la existencia en el CPC de una ROM externa de primer plano con la dirección 0. Para ello se genera un OUT &DF00,0. Si el sistema operativo determina que algo 'se mueve' aquí, se inicializa la ROM externa tomando así el control sobre la máquina. Pero en caso de que en el port de expansión no se halle ROM externa alguna con esta dirección de selección de ROM, se inicializa la ON-Board-ROM, el BASIC como ROM 0. Tras esta inicialización, el programa en primer plano ha de activar las eventualmente existentes ROMs de fondo. Para ello se pregunta a cada dirección de selección de ROM, y a partir de la dirección de ROM 7, si existe una tal ROM de fondo. En caso de que exista, es automáticamente inicializada.

Si al conectar el ordenador ha tomado el control sobre el CPC una ROM distinta a la ROM de primer plano incorporada, le está permitido reservarse una zona de la RAM como memoria de trabajo. Puede, para ello, rebajar por ejemplo el límite superior de la RAM -es decir el HIMEM-. En la reserva final de memoria para el AMSDOS se desplaza también la memoria ocupada por el AMSDOS. En este caso, la RAM de AMSDOS podría comenzar, por ejemplo, en &A000.

Esta flexibilidad del ordenador tiene, por supuesto, su precio. Todas las indicaciones de direcciones realizadas en lo sucesivo en la zona de a partir de &A700 no pueden verse en absoluto. Pueden hallarse, en las circunstancias descritas, en cualquier otra zona. Así que si se desean estructurar del modo más flexible posible programas con acceso directo a la RAM de AMSDOS, no se debería acceder jamás indirectamente a las variables. Ha de tomarse el comienzo efectivo de la RAM de AMSDOS y acceder a las variables mediante el offset al principio de la memoria.

Lo que suena bastante complicado en la descripción, se aclarará en un ejemplo posterior. Pero de momento ha de indicarse otra peculiaridad del floppy en relación con la RAM de AMSDOS. Y es que no sólo existe la zona ya descrita de 1 K, en la que se aloja la mayoría de las variables del

sistema. En la zona de &BE40 a &BE7F se utilizan otros 64 bytes para el floppy. Esta zona no es móvil, permanece siempre fija.

Esta zona es algo peligrosa. La pila del procesador suele ser ocupada en sentido descendente a partir de &C000. En programas que no gestionen correctamente dicha pila o stack, o bien precisen una pila muy grande -programación recursiva, p.ej.-, puede ocurrir que debido a un exceso de PUSH o de CALL la pila se haya tan grande que sobrepase la zona de memoria del floppy. En el primer caso deberá corregirse el programa; en el segundo, cambiar la situación del stack. Pasemos a la lista de los 64 bytes, o al menos de lo que quedó claro.

Lista de referencia RAM de sistema OBE40H hasta OBE7FH

BE40, BE41	Vector a Disk Parameter Header Drive A
BE42, BE43	Vector a Disk Parameter Block Drive A
BE44, BE45	tiempo de espera para alcanzar la velocidad de régimen del motor del Drive tras MOTOR ON
BE46, BE47	tiempo de rotación remanente del motor del Floppy tras el último acceso
BE48	valor para bucle de espera al formatear pista
BE49, BE4A	valor para bucle de retardo largo
BE4B	número de bytes en lectura estado de interrupción
BE4C, BE52	Buffer para bytes de fase de resultados de FDC
BE53	Drive(HS/US)
BE54	pista
BE55	Record
BE56	Drive(HS/US)
BE57	pista
BE58	sector
BE59	número de Records/Pista (máscara de bloque +1)
BE5A	Drive(HS/US)

BE5B	pista
BE5C	Record
BE5D	
BE5E	leer/escribir sector de Flags
BE5F	Flag de motor On/Off
BE60, BE61	Vector a Buffer de I/O de Record
BE62, BE63	Vector a Buffer de I/O de Record
BE64, BE65	memoria intermedia de Stack
BE66	número de intentos de lectura
BE67, BE6C	Tick Block
BE67, BE68	Ticker Chain, encadenamiento lista de Ticker
BE69, BE6A	Ticker Count, número de Ticker, para poner en marcha el Event
BE6B, BE6C	Reload Count
BE6D, BE73	Event-Block
BE6D, BE6E	Event Chain, encadenamiento de Events
BE6F	Event Count
BE70	Event Class
BE71, BE72	dirección de la rutina de Event
BE73	ROM-Select de la dirección FAR para la rutina de Event
BE74	Buffer para número de pista deseado
BE75	Buffer para código objeto al FDC
BE76, BE77	Vector a Buffer de I/O de sector
BE78	Flag para mensajes de error del controlador ON/OFF
BE79-BE7C	no utilizado bajo AMSDOS en la versión estudiada
BE7D, BE7E	memoria IY, Lowaddress Memory Pool para Floppy
BE7F	vector para la manipulación de las rutinas de disco (DISC OUT OPEN etc.), normalmente &C9, RETURN

Hay que hacer algunas observaciones respecto a estas direcciones. Dado que, como ya se dijo, son absolutas, es decir, no varian su posición en la memoria, pueden modificarse sus contenidos mediante sencillos POKE. En cualquier caso, algunas variables son únicamente modificadas

durante la ejecución de rutinas de disco de modo que no pueden ser modificadas con pleno sentido desde el BASIC. Pero en otros sitios pueden hacerse modificaciones esenciales con pocos POKE.

Asigne, mediante POKE, el valor 1 en la posición de memoria &BE45. Introduzca finalmente la orden CAT. El resultado es asombroso. Si tiene paciencia suficiente, habrá obtenido un CATALOGO normal y corriente. Lo que si es verdad es que ha tardado algo más de lo usual. Los valores en las posiciones de memoria &BE44 y &BE45 determinan el tiempo de espera tras encender el motor. Normalmente se espera alrededor de un segundo; nosotros hemos elevado drásticamente este valor. Puede conseguir otra manipulación de tiempos de espera, práctica en algunos casos, mediante POKEs en las posiciones de memoria &BE46/&BE47. Los valores de estas posiciones determinan el tiempo de rotación remanente de los motores de floppy tras el último acceso al disco. Puede ocurrir que en la lectura y elaboración de datos de un fichero transcurra exactamente el tiempo de espera. Para el siguiente acceso se vuelve a encender de nuevo el motor, pero a la vez se espera de nuevo un segundo para alcanzar la velocidad de régimen. Si el tiempo de rotación remanente se alarga de tal modo que el motor no se apague entre accesos, puede alcanzarse un claro incremento de la velocidad de elaboración de datos. Pruébelo.

También resultan bastante interesantes los bytes de las direcciones &BE4C a &BE52. Aquí se disponen los datos de la fase de resultados tras todas las operaciones del FDC. Puede interpretarlos usted mismo, lo que resulta esencial en caso de desconexión de los mensajes de error del control.

Hemos llegado por fin a la ya mencionada posición de memoria &HE7B. En caso de que le asignemos un valor de &FF, los mensajes de error del FDC no aparecerán en pantalla. Ante ello debe reaccionar frente a los errores según los bytes de la fase de resultados.

Muy importantes para el acceso a la zona deslizable de RAM de 1 K del AMSDOS resultan las posiciones &BE7D y &BE7E. Aquí se anota la dirección de inicio de la zona de 1 K.

Normalmente se halla aquí la dirección &A700. Pero si, por algún motivo, varía usted el AMSDOS-RAM-start, hallará en estas posiciones la dirección de inicio correcta.

La posición de memoria &BE7F será de gran interés para los más especialistas. Suele hallarse el valor &C9. Este es el código de OP del Z80 para RETURN. Cada acceso a la rutina 'patchead' CAS pasa a través de este return. Con ello tenemos una manera sencilla de modificación tanto antes como después de la ejecución de las rutinas.

De hecho cabría la posibilidad de volver a parchear la rutina patchead CAS, es decir proveerla de las direcciones de sus propias rutinas, pero este procedimiento es bastante aparatoso.

Y ahora llegamos a la parte más voluminosa en esencia de la RAM del sistema. En las siguientes exposiciones hemos supuesto que no hay ninguna ampliación conectada. Nos referimos así a las direcciones en la zona a partir de A700.

Aquellos de ustedes que sean especialistas en CP/M tendrán que investigar un poco por su cuenta, la organización de la RAM precisada por el BIOS es distinta. Así, los FDC están en las direcciones estándar de CP/M (&5C hasta &7C). Los disc parameter header se hallan a partir de &AE58, los disc parameter blocks a partir de &ADDB.

Lista de referencia de la RAM de sistema 0A700H a 0AA80H

A700	Drive anunciado
A701	User anunciado
A702	Drive activo
A703,A704	indicador a Drive activo según Disc Parameter Cabecera (a910/a920)
A705	Flag, indica si OPEN activo en Drive anunciado

A706-A707 memoria intermedia para Stack-Pointer para todas las rutinas lógicas

A708-A72B File Control Block ampliado para OPENIN

A709-A728 Buffer de File Control Block (FCB) de OPENIN
A70B Flag para OPENIN
ff = ningún OPENIN activo
00 = OPENIN en drive A
01 = OPENIN en drive B

A709 número de usuario para OPENIN

A70A-A714 nombre de fichero para OPENIN; 8 caracteres de nombre de fichero, 3 caracteres de extensión
&00 primera entrada, si no, número de extensión

A715 &00

A716 &00

A717 &00

A718 número de Records de esta extensión

A719-A728 número de bloques de esta extensión

A729-A72B número de Records leídos hasta ahora en INPUT

A72C-A74F File Control Block ampliado para OPENOUT

A750-A799 Buffer de File Control Block (FCB) de OPENOUT
A72C Flag para OPENOUT
ff = ningún OPENOUT activo
00 = OPENOUT en Drive A
01 = OPENOUT en Drive B

A72D número de usuario para OPENOUT

A72E-A738 nombre de fichero para OPENOUT 8 caracteres de Filename,
3 de extensión rellenos con espacios
&00 primera entrada, si no, número de extensión

A739 &00

A73A &00

A73B &00

A73C número de Records de esta extensión

A73D-A74C n mero de bloques de esta extensi n
A74D-A7AF n mero de Records leidos hasta ahora en OUTPUT

A750-A799 cabecera de fichero OPENIN

A750 1 = Disc in Char
 2 = Disc in Direct
A751-A752 Vector al comienzo del Buffer de 2 K de OPENIN
A753-A754 Vector al car cter actual en Buffer de OPENIN
A755 n mero de usuario del fichero, parte constituyente de su nombre de fichero
A756-A764 nombre de fichero para cabecera, rellenado con ceros
A755 n mero de bloque
A766 Last Block
A767 tipo de fichero para fichero de entrada(INPUT)
A768-A769 Data Length
A76A-A76B Data Location
A76C First Block
A76D-A76E Logical Length
A76F-A770 Entry Adress
A771-A794 User Fields (libre para el usuario)
A795-A797 contador de 3 bytes n mero caracteres leidos
A798-A799 suma de prueba de 2 bytes extendida la cabecera de fichero OPENIN (a755-a797)

A79A-A7E3 cabecera de fichero OPENOUT

A79A 1 = Disc Out Char
 2 = Disc Out Direct
A79B-A79C Vector al comienzo del Buffer de 2 K de OPENOUT
A79D-A79E Vector al car cter actual en el Buffer de OPENOUT
A79F n mero de usuario de fichero, parte de su nombre

A7A0-A7AE	nombre de fichero para cabecera, rellenado con ceros
A7AF	número de bloque
A7B0	Last Block
A7B1	tipo de fichero para fichero de salida(OUTPUT)
A7B2-A7B3	Data Length
A7B4-A7B5	Data Location
A7B6	First Block
A7B7-A7B8	Logical Length
A7B9-A7BA	Entry Adress
A7BD-A7BE	longitud bloque de datos con Disc Out Direct
A7BF-A7C0	dirección de Entry con Disc Out Direct
A7C1-A7DE	User Fields (libre para el usuario)
A7DF-A7E1	contador de 3 bytes número de caracteres escritos
A7E2-A7E3	suma de prueba de 2 bytes extendida a la cabecera de fichero OPENOUT (a79f-a7e1)
A7E4-A8E3	Buffer Temporal/Record-Buffer Este Buffer se usa, tanto como Record-Buffer como para comprobar y expandir el nombre de fichero dado.
A874-ABBA	Buffer para Tape-Vectores (Vectores del Cassette) originalmente dipuestos, vuelven a ser puestos tras BC77.. con el comando 'ITAPE'
A88B-A88D	dirección Far para vectores de Cassette con parches preciso para el RST 4 incorporado. Indica hCD30 en la ROM 7.
A890-A8A8	Extended Disk Parameter Block Drive A
A890,A891	STP Records por pista (36)
A892	BSH Block Shift (3)

A893	BLM	Block Mask (7)
A894	EXM	Extend Mask (0)
A895, A896	DSM	número de bloque màximo (170)
A897, A898	DRM	màximo de entradas en directorio 1 (63)
A899, A89A	ALO, 1	magnitud de indice (hC000), en binario; supone 2 bloques
A89B, A89C	CKS	número de entradas a comprobar en el indice (0010) 16 asientos
A89D, A89E	OFF	Offset de pista (2) pistas de sistema ocupadas
A89F-A8A8	paràmetros FDC	
A89F	FSC	primer sector de cada pista (h41)
A8A0	PST	sectores físicos por pista (9)
A8A1	GPS	longitud GAP 3 en R/W de sector (h2a)
A8A2	GPT	longitud GAP 3 al formatear pista (h52)
A8A3	FLB	Byte "relleno" al formatear pista (he5)
A8A4	BPS	Bytes/Sector comportan 512 bytes
A8A5	RPS	número de Records/Sector (4)
A8A6	Buffer	para pista actual
A8A7		buscar Flag de pista 0, Read/Write recalibrado
A8A8	Flag,	indica si ha de alcanzarse Login en cada acceso
A8A9-A8B8	CSA	16 bytes para sumas de prueba
A8B9-A8CE	ALT	22 bytes de Allocation Table disposiciòn de bloques Drive A
A8D0-A8E8		Extended Disk Parameter Block Drive B disposiciòn como DPB Drive a
A8E9-A8F8	CSA	16 bytes para sumas de prueba

A8F9-A90E	ALT 22 bytes de Allocation Table (Tabla de asignaciòn) disposiciòn de bloques Drive B
A910-A91F	Disk Parameter Header Drive A
A910-A911	XLT factor Skew de conversiòn de tabla (no utilizado)
A912-A913	TRACK memoria BIOS pista actual. Cuidado, es utilizado por el AMSDOS como DIRNUM
A914-A915	SECTOR memoria BIOS para sector actual
A916-A917	DIRNUM memoria BIOS para nùmero DIR actual
A918,A919	DIRBUF puntero a I/O Buffer de 128 bytes(a930)
A91A,A91B	DBP puntero a DBP Drive A (a890)
A91C,A91D	CSV puntero a memoria para la formaciòn de la suma de prueba (a8a9)
A91E,A91F	ALV puntero a tabla de disposiciòn (a8b9)
A920-A92F	Disk Parameter Header Drive B
A920-A921	XLT factor Skew de conversiòn de tabla (no utilizado)
A922-A923	TRACK memoria BIOS pista actual. Cuidado, es utilizado por el AMSDOS como DIRNUM
A924-A925	SECTOR memoria BIOS para sector actual
A926-A927	DIRNUM memoria BIOS para nùmero DIR actual
A928,A929	DIRBUF puntero a I/O Buffer de 128 bytes(a930)
A92A,A92B	DBP puntero a DBP Drive B (a8d0)
A92C,A92D	CSV puntero a memoria para la formaciòn de la suma de prueba (a8a9)
A92E,A92F	ALV puntero a tabla de disposiciòn (a8f9)
A930-A9AF	DIRREC Buffer de 128 bytes para un Record de directorio.

Transferido aquí desde
el sector DIR

A9B0-ABA5 SECBUF Buffer para la transferencia física de
datos
del y hacia el Floppy.

Seguramente estará usted tentado de comprobar los efectos de la manipulación de algunas posiciones de memoria. Bueno, que se sepa no puede estropearse por ello el ordenador. En cualquier caso, debería evitar realizar sus primeros intentos con su disco preferido insertado y no protegido frente a escritura. Una vez borrados los datos en el disco resulta demasiado tarde para el largamente postergado backup.

Si observamos con detalle la zona de memoria, puede distinguirse claramente la frontera entre distintas zonas individuales. Hemos denominado estas zonas con nombres que no serán conocidos para todos los lectores. Hemos tomado estos conceptos del CP/M dado que muchas zonas de datos son idénticas en sus funciones a aquellas en el CP/M. Las zonas Disk Parameter Header y Disk Parameter Block están escritas (casi) exactamente igual en el CP/M. Todo ordenador con CP/M tiene DPBs y DPHs.

También se utilizan en CP/M los FDCs, los File Control Blocks. En cualquier caso, algunas zonas han sido ampliadas en el AMSDOS. Un DBP de CP/M estándar contiene únicamente los 15 bytes STP hasta OFF, las ampliaciones son específicas del AMSDOS y no pueden ser traducibles al CP/M de otros ordenadores; de hecho ni siquiera al CP/M del CPC. También son específicos del AMSDOS los Fileheader para OPENOUT y OPENIN.

4.2 El listado de la ROM del AMSDOS

En las páginas siguientes no hallarás ningún listado completo de ROM. En lugar de ello, "sólo" se imprimen los comentarios a las distintas órdenes, sin que dispongan, posiblemente, de sentido inmediato. Pero en unión con un programa monitor, o bien un desensamblador, puede ampliar sin más nuestros comentarios incorporando los mnemónicos. Dado que está dispuesto y deseoso de manejar con soltura un sistema operativo, estamos seguros de que estos "comentarios" a nuestros comentarios no le resultarán pesados.

Como ya se advirtió, el AMSDOS no ocupa siquiera la mitad de los 16 K disponibles de la ROM. 8 K completos se ocupan con partes del LOGO suministrado con el Floppy. Ello reduce a la mitad la zona disponible.

Esta zona de LOGO no se imprime en el listado por varios motivos. En primer lugar tiene usted en sus manos un libro sobre el Floppy y no sobre LOGO. Por otro lado, la parte de LOGO de la ROM sólo es una exigua porción del intérprete LOGO total. Los 32 K de este lenguaje de programación, muy interesante por otra parte, los hallarás en el disco de sistema del CPC. Si imprimiéramos y comentáramos la parte de LOGO de la ROM, serviría de bien poco por ser una porción del intérprete completo.

Pero ni siquiera los 8 K restantes son completamente ocupados por el AMSDOS. Los 1024 bytes en la zona de &DC00 a &DFFF no se usan en absoluto. Posiblemente exista esta zona para futuras ampliaciones del AMSDOS. Nos quedan ya sólo 7 K, de los que tampoco dispone totalmente el AMSDOS. También se integran en esta zona partes del CP/M. Para ello utilizan

CP/M y AMSDOS muchas rutinas de la ROM de forma conjunta, mientras que otras son utilizadas de forma exclusiva por el AMSDOS o por el CPM. Se hallan así incorporados en la ROM dos programas completos para dos interfaces serie, que pueden ser asignados bajo CP/M a los distintos dispositivos de muy diversas formas a través del byte de I/O.

Un vistazo al listado revela que, tras restar todas las zonas de memoria exclusivamente utilizadas por el CP/M, quedan apenas 6 K para el AMSDOS. Pero 6 K de los que entran pocos en un kilo. Véalo usted mismo.

* * * * Prefijo para ROM de CP/M
C000 ROM type, background ROM
C001 ROM mark number
C002 ROM version number
C003 ROM modification level

* * * * Dirección de la tabla de comandos
C004

* * * * Bloque de saltos a órdenes AMSDOS
C006 CPM ROM
C009 CPM
C00C DISC
C00F DISCIN
C012 DISCOUT
C015 TAPE
C018 TAPEIN
C01B TAPEOUT
C01E A:
C021 B:
C024 DRIVE
C027 USER
C02A DIR
C02D ERA
C030 REN

* * * * Bloque de saltos a órdenes del controlador de disco
C033 ^81 Mensajes de error enable/disable
C036 ^82 Indicar datos unidad disco
C039 ^83 Determinar formato disco
C03C ^84 Leer sector
C03F ^85 Escribir sector
C042 ^86 Formatear pista
C045 ^87 Buscar pista
C048 ^88 Determinar estado unidad discos
C04B ^89 Fijar número intentos lectura

* * * * Bloque de saltos introducción en CP/M
C04E
C051

* * * * Bloque de saltos rutinas de I/O serie para CP/M
C054 Inicialización completa SIO & 8253
C057
C05A Chan. A buffer RX lleno?
C05D Chan. A recoger un carácter
C060 Chan. A buffer TX lleno?
C063 Chan. A enviar un carácter
C066 Chan. B buffer RX lleno?
C069 Chan. B recoger un carácter
C06C Chan. B buffer TX lleno?
C06F Chan. B enviar un carácter

*** * * * Tabla de órdenes DOS****C072**
C0B3*** * * * Órdenes controlador de disco****COB6**
COBF Indicativo fin de la tabla*** * * * Rescatar vector de interrupción & dirección de port GA****COC0** Vector INT (RST7)
COC3 anotar en RAM
COC6 Código objeto JMP
COC8 para handler de CALL**COCB****COCC****COCF** Ofrecer INT para uso**CODO** del juego alternativo de registros
COD1 Rescatar GA dirección de port y configuración de la ROM
COD5 Comutar juego de registros a posición anterior**COD6****COD9***** * * *****CODB****COEB** para utilizar el antiguo juego de registros**COE9****COEA****COEB** Volver a tomar dirección de port GA y configuración de ROM**COEF** Borrar carry**COFO****COF1****COF2****COF5** Dirección del salto en AD33**COFB** Volver a liberar interrupción**COF9***** * * * Handler 'CALL AD33', se conserva juego antiguo registros****COFA** Ofrecer interrupciones para**COFB** trabajar con el juego**COFC** de registros alternativo**COFD** Anotar hl**C100** Dirección de return a hl**C101** Almacenar puntero de la pila (stackpointer)**C105** e inicializar todos los**C108** registros en el stack inicializado**C109****C10B****C10D** (Dirección de port gate array)**C111****C112** Tomar dirección tras 'CALL AD33' y efectuar CALL**C115** podría volver a ser aceptado**C116** Variar registros**C117**

C11B Se ha modificado eventualmente
C11C Reorientar vector de INT
C11F
C122 Recoger registros
C124
C126
C127 hl no fue objeto de PUSH
C12A
C12B Comutar al juego de registros std.
C12C Restaurar el antiguo stackpointer
C130
C131 Rutina deseada ejecutada

* * * * Handler 'CALL AD33', no se rescata juego antiguo registros
C132 Ofrecer INT
C133 Juego de registros alternativo
C134
C135 Dirección de retorno del stack
C136 Almacenar stackpointer
C13A Inicializar stack
C13D Tomar dirección tras 'CALL AD33' y efectuar JP
C140 Podria volver a ser aceptado
C141 Variar registros
C142 Reorientar vector de INT
C145
C148 Comutar juego de registros std.
C149 Restaurar el antiguo stackpointer
C14D
C14E rutina deseada ejecutada

* * * * JP (salto) a la dirección tras 'CALL AD33'
(system interrupt vector)
C153 Orientar vector de INT
C157 Dirección base para disc-RAM
C15B Secuencia de bytes tras CALL AD33
C15C recoger en de
C15D
C15E al stack
C15F
C161
C162 y saltar mediante RET

* * * *

C163
C166 Salto al vector de INT

* * * *

C168 Anotar hl
C16B Dirección de RET a hl
C16C
C16D Incrementar en dos, o sea a dirección dada
C16E
C16F Intercambiar con dirección de RET

C170 Valor original al stack
C171 Restaurar hl
C174

* * * * Anotar vector de INT del sistema, JP (DE)
C177
C17E

* * * * Bloque de salto BIOS, en RAM bajo CP/M a partir de 0AD00H
C17F COLD BOOT
C182 WARM BOOT
C185 CONSOLE STATUS
C188 CONSOLE INPUT
C18B CONSOLE OUTPUT
C18E PRINTER OUTPUT
C191 PUNCHER
C194 READER
C197 TRACK 0
C19A SELECT DRIVE
C19D SELECT TRACK
C1A0 SELECT SECTOR
C1A3 INSTALL BUFFER
C1A6 READ SECTOR
C1A9 WRITE SECTOR
C1AC PRINTER STATUS
C1AF TRADUCIR NUMERO DE SECTOR

* * * * CPM COLD BOOT
C1B2 KL Curr Selection
C1B5 ROM selección
C1B6 Dirección de entry point
C1B9 Mc START PROGRAM

* * * * CPM ROM
C1BC
C1BE KL Curr Selection
C1C1 Prueba de nulidad dirección ROM select
C1C2 => LK 1 abierto a contr. board , CP/M
C1C4 Dirección himem
C1C6 Dirección lomem
C1C7 hl = himem
C1CA Rebañado en h0400
C1CB Anotar nueva himem
C1CC
C1CD
C1CE iy = himem + 1
C1DO Inicializar FDC y event
C1D3 Patch de vectores de cassette
C1D6 Transferir los nuevos
C1D7 valores para lomem e
C1DB himem a KL START PROGRAM

C1DA Indicativo de inicialización OK
C1DB

* * * * ENTRY arranque en frío de CP/M
C1DC Inicializar stack
C1DF
C1E6
C1E9 Borra (de) hasta (de+bc)
C1EC
C1EF
C1F0 Fijación estándar de IO-byte
C1F2
C1F5 Drive y user
C1F6
C1F9 Tabla de salto de comandos del controlador &81-&89
C1FC Copiar a BE80
C1FF En total h3F bytes
C202 Transferir
C204 Salvar vector INT & dirección GA de port
C207 Inicializar FDC y event
C20A Número de sector
C20C Track y drive
C20F Dirección del buffer
C212 Leer sector
C215 => sector leído vacío?
C218 Apareció error
C21A
C21E
C221 Vector INT a C163. JP(DE)

* * * * Error en la lectura del sector de BOOT
C224 Msg. 15 'Failed to load Boot sector'
C226 Preguntar 'CHAN., IGN. or RETRY'
C229

* * * * Cargar CP/M CCP y BDOS de disco, Boot en caliente
C22B
C22E
C231 b=contador de sector, c=número de sector
C234 Sector y drive
C237 Anotar dirección del buffer
C238 Leer cifra deseada de sectores
C23B Comienzo buffer de nuevo a hl
C23C Comprueba si el sector leído está vacío
C23F Apareció error
C241
C24B
C24C Longitud de sector
C24F
C251 Dirección de buffer
C252 b=contador de sector, c=número de sector
C255 Sector y drive
C258 Leer número deseado de sectores

C25B Apareció error
C25D
C266
C267 Código objeto para JP
C269
C26C BDOS-entry, 8F00
C26F JP-code
C272
C275
C276 BDOS-entry, AD00
C279 Tabla de los vectores CP/M
C27C 51 bytes de longitud
C27F hacia AD00
C281 Usuario y drive
C284
C285 Desenmascarar drive
C287 Drive 0 ó 1 ?
C289 Si es 0 ó 1 =>
C28B En caso contrario poner a 0
C28D
C28E
C28F Vector INT a C163. JP(de)

* * * * Error en la carga de CP/M
C292 Msg. 14 'Failed to load CP/M'
C294 Preguntar 'CHAN., IGN. or RETRY'
C297 WARM BOOT

* * * *carga sectores corrientes, núm. b, sect. c, pista d
C299 Cargar sector
C29C Apareció error
C29D Número de sector al acumulador
C29E Sector siguiente
C29F El último sector fue el número h49 ?
C2A1 => no fue el h49
C2A3 Sector h41
C2A5 A pista siguiente
C2A6 Incrementar indicador buffer en 2 pages
C2A7
C2A8 Leídos todos los sectores ?
C2AA Indicativo todo OK
C2AB

* * * * comprueba si el sector leído está vacío
C2AC Dirección del buffer
C2AD 512 bytes
C2B0 Primer carácter del buffer al acumulador
(indicador buffer)
C2B1
C2B2 Incrementar indicador
C2B3 Indicativo OK
C2B4 Si distinto, entonces =>

C2B6 Bucle sobre 256 bytes
C2B8 Por dos = un sector
C2B9
C2BB Apagar carry
C2BC repetir indicador buffer
C2BD

* * * * WARM BOOT
C2BE 'JP C22B'
C2C1

* * * * CONSOLE INPUT
C2C3 (h1)=> consola (teclado) en coordinacin
C2C6

* * * * CONSOLE OUTPUT
C2C8 (h1)=> Coordinacin consola out
C2CB

* * * * PRINTER STATUS
C2CD (h1)=> Coordinacin list device status
C2D0

* * * * PRINTER OUTPUT
C2D2 (h1)=> Coordinacin list device output
C2D5

* * * * PUNCHER
C2D7 (h1)=> Coordinacin puncher
C2DA

* * * * READER
C2DC (h1)=> Coordinacin reader
C2DF

* * * * CONSOLE STATUS
C2E1 (h1)=> Coordinacin console status
C2E4 'JP C46A'C2DA Coordinacin a travs de I/O-byte
C2E7

* * * * BUSCAR TRACK 0
C2E9 'JP C51F'
C2EC
C2F1

* * * * SELECT DRIVE
C2F2 'JP C4F0'
C2F5

* * * * READ SECTOR
C2F7 'JP C54C
C2FA

* * * * WRITE SECTOR

C2FC 'JP C52E'

C2FF

* * * *

C301

C312

* * * * Comprobar estado teclado, carácter disponible?

C313

C31A

C31B TXT CURSOR ON

C31E KM READ CHAR, un carácter de teclado

C321 Si hay carácter disponible, KM RETURN CHAR

C324 Offh=> carácter disponible, 00 ningún carácter

C325

* * * * Console input, recoger un carácter de teclado

C326 Keyboard modus flag

C329

C32A Comprobar flag

C32B => Keyboard mode 'INPUT'

C32D Keyboard mode 'INKEY', KM READ CHAR

C330 => no recibir carácter alguno

C332

C33D

* * * *

C33E Keyboard mode a 'INPUT'

C33F

C347

* * * * Console input, esperar un carácter de teclado

C348

C34C

C34D TXT CURSOR ON

C350

C352 . KM WAIT CHAR, esperar carácter

* * * * High speed reader como reader, no implementado

C355 EOF

C357

* * * * Status CTR como printer, high speed reader como reader
C358

* * * * High speed puncher como puncher device

C35A

* * * * CRT-device, dar salida a pantalla a un carácter

C35B

C35E

```

C35F
C360      TXT CURSOR OFF
C363
C365      Caràcter a salir al acumulador
C366      TXT OUTPUT
C369      Caràcter vacío (SPACE)
C36B      => ningún còdigo de control
C36C      TXT GET CURSOR
C36F      TXT VALIDATE
C372
C373      TXT PLACE CURSOR
C376      TXT REMOVE CURSOR

* * * * Line printer status, comprueba si centronics busy
C379      MC BUSY PRINTER, carry en caso de busy
C37C      carry en caso de que no busy
C37D      Offh=> no busy, 00 si no busy
C37E

* * * * Line printer output, un caràcter a impresora
C37F      Caràcter al acumulador
C380      MC PRINT CHAR, dar salida
C383      Envio fructífero de caràcter
C384      Printer busy, comprobar teclado
C387      Nuevo intento

* * * * inicializar I/O serie (hl)=> tabla de paràmetros
C389
C38A      SIO channel A/registro de control
C38D      Memoria para WR-reg. 5, canal A
C390      Channel reset, inicializa canal A
C393
C394      SIO channel B/registro de control
C395      Memoria para WR-reg. 5, canal B
C396      Channel reset, inicializa canal B
C399      Modus timer 0 del 8253
C39B      Byte low para dirección del port timer 0
C39D      Fijar velocidad de transferencia de envío del canal A
C3A0      Modus timer 1 del 8253
C3A2      Byte low para dirección del port timer 1
C3A3      Fijar velocidad de transferencia de recepción del canal A
C3A6      Modus timer 2 del 8253
C3A8      Byte low para dirección del port timer 2
C3A9      Fijar vel. transf. de envío y recepción del canal B
C3AC
C3AD

* * * * inicializar generador vel. transf. 8253; (hl)=>valores low-high
C3AE      Dirección del port palabra de control:8253
C3B1      Dar salida a 8253 de la palabra de control
C3B3      Low-byte de dirección del port del timer elegido
C3B4      Low-byte del valor del timer
C3B5

```

C3B6 Cargar en el timer
C3B8 High-byte del valor del timer
C3B9
C3BA Cargar en el timer
C3BC

* * * * inicializar canal SIO en (BC)
C3BD Fijar canal de código objeto en valor anterior
C3BF Dar salida hacia SIO
C3C1 Elegir write-register (registro de escritura) 4
C3C3
C3C5 Apunte en tabla para
C3C6 paridad, bytes de paro y clock-mode
C3C7 dar salida hacia SIO
C3C9 Elegir write-register 5
C3CB
C3CD Apunte en tabla para protocolo y bits/char
C3CE Anotar bits/char en (de) (ADC6/ADC7)
C3CF
C3D0 y dar salida hacia SIO
C3D2 Elegir write-register 3
C3D4
C3D6 Valor en tabla para protocolo y bits/char
C3D7
C3D8 dar salida hacia SIO, parámetro de recepción, RX
C3DA

* * * * canal A, buffer RX lleno?
C3DB Canal A SIO, registro de control
C3DE (hl)=> contenido del registro write 5, canal A
C3E1

* * * * canal B, buffer RX lleno?
C3E3 Canal B SIO, registro de control
C3E6 (hl)=> contenido del registro write 5, canal B
C3E9 Registro read 0 del canal elegido
C3EB Bit 0, carácter RX listo?
C3EC
C3ED => existe un carácter
C3EE Fijar bit DTR
C3F1 Leer registro read 0
C3F3 Bit 0, carácter RX listo?
C3F4
C3F5 Fijar bit DTR en valor anterior

* * * * canal A SIO recoger un carácter
C3F7 Canal A SIO, registro de control
C3FA (hl)=> contenido del registro write 5, canal A
C3FD

* * * * canal B SIO recoger un carácter
C3FF Canal B SIO, registro de control

(C402 (h1)=> Contenido del registro write 5, canal B
C405 Leer registro read 0
C407 Caràcter RX disponible?
C408 => recibir un caràcter
C40A Fijar bit DTR
C40D Comprobar teclado
C410 Entrar control Z como fin?
C412 => volver a fijar bit de DTR, RET
C414 Leer registro read 0
C416 Caràcter RX disponible?
C417 => no disponible
C419 Volver a fijar bit de DTR
C41C Dirección de port del registro de datos
C41D Leer caràcter recibido
C41F

* * * * Volver a fijar bit de DTR, posibilitar recepción
C420 Borrar bit 7, es relevante
C422

* * * * Fijar bit de DTR, imposibilitar recepción
C424 Fijar bit 7
C426
C427
C428 Registro de Write 5
C42A elegirlo
C42C (h1)=> contenido del registro WR 5
C42D Cegar bit 7
C42F Según el salto, bit 7 activado/desactivado
C430 Escribir en write-register 5
C432
C434

* * * * Comprueba si TX-buffer canal A vacío
C435 SIO canal A, registro de control
C438

* * * * Comprueba si TX-buffer canal B vacío
C43A SIO canal B, registro de control
C43D Leer registro read 0
C43F Desenmascarar TX-empty-bit
C441 => buffer no lleno
C442 Indicativo de buffer no lleno
C443
C444

* * * * Enviar un caràcter a través del canal A
C445 Caràcter al que hay que dar salida al acumulador
C446 SIO canal A, registro de control
C449

* * * * Enviar un carácter a través del canal B
C44B Caràcter al que hay que dar salida al acumulador
C44C SIO channel B, registro de control
C44F Almacenar carácter
C450 Preguntar al teclado
C453 Buffer de emisión vacío?
C456 => aún no vacío
C458 Caràcter de nuevo al acumulador
C459 Dirección de port del registro de datos
C45A Escribir carácter en SIO
C45C

* * * * Determinar READER-status mediante byte de I/O
C45D Tabla de READER-status
C460

* * * * READER-input mediante byte de I/O
C462 Tabla de READER-input
C465

* * * * PRINTER-OUTPUT mediante byte de I/O
C467 Tabla de PRINTER-output

* * * * Determinar I/O-device con I/O-byte, (h1) =>tabla asignación
C46A Número de bucles para los cuatro dispositivos
C46B (h1)=> primera asignación
C46C I/O-byte, usualmente &81
C46F
C470 (b) por I/O-byte hacia la izquierda
C472 Aislar byte relevante
C474
C476 Da offset en tabla de asignación
C477 Sumar a start
C478 Dirección de la rutina de I/O a de
C479
C47B
C47C Salto indirecto a rutina I/O

* * * * CONSOLE STATUS
C47D
C47E JP OC3D8H, carácter SIO canal A disponible?
C480 Caràcter de teclado disponible?
C482 READER-status mediante byte de I/O
C484 JP OC3E3H, carácter SIO canal B disponible?

* * * * CONSOLE INPUT
C486
C487 JP OC3F7H, recoger carácter de SIO canal A
C489 Recoger carácter del teclado
C48B Leer carácter de READER mediante byte de I/O
C48D JP OC3FFH, recoger carácter de SIO canal B

*** * * * CONSOLE OUTPUT**

C48F

C490 JP OC445H, enviar caràcter a travès de SIO canal A
C492 Dar salida a pantalla al caràcter
C494 PRINTER-OUTPUT mediante byte de I/O
C496 JP OC44BH, enviar caràcter a travès de SIO canal B

*** * * * PRINTER STATUS**

C498

C499 JP OC435H, buffer de envio de canal A vacio?
C49B No implementado
C49D Chequea centronics busy
C49F JP OC43AH, buffer de envio de canal B vacio?

*** * * * PRINTER OUTPUT**

C4A1

C4A2 JP OC445H, enviar caràcter a travès de SIO canal A
C4A4 Dar salida a pantalla al caràcter
C4A6 Da salida a caràcter al port de centronics
C4AB JP OC44BH, enviar caràcter a travès de SIO canal B

*** * * * PUNCHER**

C4AA

C4AB JP OC445H, enviar caràcter a travès de SIO canal A
C4AD No implementado, RET
C4AF JP OC44BH, enviar caràcter a travès de SIO canal B
C4B1 Dar salida a pantalla al caràcter

*** * * * READER Status**

C4B3

C4B4 JP OC3D8H, caràcter SIO canal A disponible?
C4B6 No implementado
C4B8 JP OC3E3H, caràcter SIO canal B disponible?
C4BA Comprueba si existe caràcter de teclado

*** * * * READER Leer caràcter**

C4BC

C4BD JP OC3F7H, recoger caràcter de SIO canal A
C4BF No implementado, recoge EOF
C4C1 JP OC3FFH, recoger caràcter de SIO canal B
C4C3 recoge caràcter del teclado

*** * * ***

C4C5 Comprueba si se pulsa CONTROL C

C4CB ENTER?

C4CA ==> no ENTER

C4CC

C4CD recoger otro caràcter del teclado

C4DO

C4D2

* * * * Comprobar si se pulsa CONTROL C

C4D3
C4D5
C4D6 Comprobar estado del teclado
C4D9
C4DA => ningún carácter disponible
C4DC Recoger carácter de teclado
C4DF Control C?
C4E1 => no control C
C4E3 Mensaje del sistema 14 ..^C
C4E5 darle salida
C4E8 Warm boot

* * * * ninguna acción, no se pulsa CONTROL C

C4EB
C4EE

* * * * no utilizado

C4EF

* * * * SELECT DRIVE

C4F0 Número de drive al acumulador
C4F1 sólo puede ser 0 ó 1
C4F3
C4F6 Número de drive demasiado alto
C4F7 Número de drive válido hasta ahora al acumulador
C4F8 Bit 0 al carry
C4F9 Salto, si drive 1 activo hasta ahora
C4FB Drive deseado a e
C4FC
C4FE Valor del bloque de parámetros de disco al acumulador
C501 si es no nulo
C502 entonces, salto
C504
C505 en caso contrario, determinar formato de disco
C508
C509
C50A Número de drive al acumulador
C50B HS/US-buffer
C50E Offset a disc parameter header drive A
C511 si se usa drive A, salto
C512
C514 en caso contrario, offset DHP para drive B
C517 hl=hl+iy, hl=> comienzo de tabla de punteros

* * * * Apuntar dirección buffer de record

C51A (bc):= record-buffer
C51E

```
* * * * buscar pista 0
C51F
C524
C525      (buffer para número de pista)
C52B

* * * * número de registro al controlador
C529
C52A      (buffer para número de registro)
C52D

* * * * Write Record (registro)
C52D
C530
C532      Drive, pista, registro de be53h a be5ah
C535      Comprobar si drive, pista y record en be53h = be5ah
C538      =>son iguales
C538      Determinar sector a partir de número de registro
C53E
C53F
C540      Transfiere registro al buffer de sector
C543      Número de registros
C544
C545      Sumar offset de sector, escribir sector
C548      => error
C549
C54B

* * * * Read Record
C54C      Borrar acumulador y
C54D      anotar +1 en máscara de bloque
C550      Determinar sector a partir de número de registro
C553      Transferir registro al buffer de registro
C556
C559

* * * * Traducir número de registro
C55A      Transfiere únicamente bc a hl
C55B
C55C

* * * * Leer sector ID, preguntar por eventuales errores
C56D      Registro de estado FDC
C560      Leer código de sector ID
C562      Dar salida al acumulador hacia FDC
C565      Unit select/head select
C566      Dar salida al acumulador hacia FDC
C569      Fase de resultados del FDC, drive ready?
```

* * * * Proporciona, del sector ID, el formato de disco
C56C Encender motor, sumar tiempo de funcionamiento del contador
C56F
C571 Cargar acumulador con valor bloque paràmetros disco 16h
C574 es el nùmero actual de pista
C575 Nùmero de intentos de lectura
C577 Dirección de la rutina 'leer sector ID'
C57A Buscar pista en d
C57D NC = error
C57E Nùmero de sector del FDC en fase de resultados ,

* * * * ^83h proporcionar formateo disco
C581 Contiene el nùmero de sector leido por el FDC
C582 Borrar acumulador
C583 Comienzo bloque paràmetros disco a h1
C586
C587
C588 Standart-disc-parameter-block en ROM
C58B 22 bytes
C58E en disc-parameter-block (DPB) actual
C590 Inicio de tabla
C591 Nùmero de sector
C592 Borrar bits 0 a 5
C594 Bit 6 activado?
C596
C597 en tal caso utilizar tabla de formato estàndar
C598 Inicio DPB de formato de datos, tabla 2
C59B Bits 6 y 7 activados?
C59D en tal caso utilizar formato de datos
C59F Inicio DPB de formato IBM, tabla 1
C5A2 Tomar los dos primeros valores de tabla
C5A3 transferirlos al disc-parameter-block
C5A4
C5A9
CSAC h1 apunta a disc-parameter-block + 5
CSAD Nùmero de bloques/disco
CSAE Transferir
CSAF los dos bytes siguientes
CSBO al disc-parameter-block actual
C5B1
C5B4
C5B7 h1 apunta a disc-parameter-block + 13
C5B8
C5B9 Transferir los 6 bytes restantes
C5BC al disk-parameter-block
C5BE
C5BF

* * * * Valores de tabla para formato IBM

C5C0 Número de registros/pista, STP
C5C2 Número de bloques/disco, DSM
C5C4 Número de pistas para el sistema operativo, OFF
C5C6 Offset de sector
C5C7 Sectores/pista
C5C8 Longitud GAP 3 read/write
C5C9 Longitud GAP 3 formateado

* * * * Valores de tabla para formato disco de datos

C5CA Número de registros/pista, STP
C5CC Número de bloques/disco, DSM
C5CE Número de pistas para el sistema operativo, OFF
C5D0 Offset de sector
C5D1 Sectores/pista
C5D2 Longitud GAP 3 read/write
C5D3 Longitud GAP 3 formateado

* * * * La tabla (7 bytes) es copiada en BE44 ...

C5D4 Tiempo máximo de funcionamiento para motor de discos
C5D6 Contador de tiempo de funcionamiento para motor de discos
C5D8
C5D9 Valores para largos bucles de demora

* * * * Tabla (2 bytes) precisa para &82

C5DB Head unload time para FDC = 32 ms
C5DC Head Load time para FDC = 16 ms

* * * * Inicializa DPHs, DPBs, FDC y event

C5DD Dirección RAM para rutinas del controlador
C5E0 Número de bytes
C5E3 Borra (de) hasta (de+bc)
C5E6 Inicializa disc parameter block/header
C5E9 Desconectar motor de drive
C5EC Tabla de parámetros de FDC
C5EF Inicializar parámetros de drive del FDC
C5F2 KL ASK CURR SELECTION
C5F5 Selección de ROM para rutina de event
C5F6 Clase de event asincrónica
C5FB Bloque de event
C5FB Dirección de la rutina de event
C5FE KL INIT EVENT
C601

* * * * ^89 fija número de intentos de lectura, número en acumulador

C603 Número de intentos de lectura
C604
C607
C60C

* * * * ^82 especificar datos de unidad
C60D RAM de FDC +4
C610
C613
C615 Registro de estado del FDC
C618 OP-code specify drive parameter
C61A darle salida hacia el FDC
C61D Tiempo de espera en milisegundos (12)
C620
C624
C625 Da como resultado hA0 = 12 ms de step rate
C627 Head unload time a bits 0 a 3
C628 darle salida hacia el FDC
C62B
C62C Head load time
C62D darle salida hacia el FDC

* * * * ^88 determinar estado unidad
C630 Rutina determinar estado de unidad
C633 => apareció algún error
C634 Cargar acumulador con estado de FDC 0
C637

* * * * Rutina determinar estado de unidad
C638 Encender motor, sumar contador para tiempo transcurrido
C63B El acumulador contiene número de drive
C63C Sense interrupt status FDC
C63F Registro de estado del FDC
C642 OP-code sense drive status
C644 Dar salida al acumulador hacia el FDC
C647 Número de drive
C648 Dar salida al acumulador hacia el FDC
C64B Leer fase de resultados

* * * * ^85 escribir sector e=drive, d=pista, c=sector, h1=I/O buffer
C64E Escribir sector OP-code
C650

* * * * ^86 formatear pista
C652 Formatear pista OP-code
C654 Encender motor, sumar contador para tiempo transcurrido
C657 Número de intentos
C659 Read/Write/format cont'd
C65C
C665

* * * * ^84 leer sector e=drive, d=pista, c=sector, h1=I/O buffer
C666 Encender motor, sumar contador para tiempo transcurrido
C669 Leer sector OP-code
C66B

* * * * Read/Write/format cont'd

C66D Buffer de I/O de 512 bytes
C670 OP-code FDC
C671 Número de sector
C672 Anotar código objeto y sector deseados
C675 Número de intentos de lectura
C676 Programar dirección de FDC
C679 Buscar pista en d, 'CALL (hl)'

* * * * Programar FDC para acción deseada
(OP-code FDC y sector deseado)

C67F Registro de estado del FDC
C682 OP-code de FDC
C683 darle salida hacia el FDC
C686 Head select/unit select, así como número de drive
C687 darles salida hacia el FDC
C68A OP-code salido al acumulador
C68B Formatear pista?
C68D Salto en caso de escritura/lectura de sector
C68F
C691 Bytes/sector, de disc-parameter-block al FDC
C694
C696 Sectores/pista, de disc-parameter-block al FDC
C699
C69B Longitud de GAP 3, de disc-parameter-block al FDC
C69E
C6A0 Byte de relleno, valor en disc-parameter-block h13
C6A3 Fase de ejecución de Read/Write/format

* * * * Llegada a lectura/escritura de un sector

C6A5 Número de pista
C6A6 Acumulador al FDC
C6A9 Número de cabezal (para unidades con dos cabezales)
C6AA Acumulador al FDC
C6AD Número de sector
C6AE Acumulador al FDC
C6B1 Bytes/sector
C6B3 de disc-parameter-block al FDC
C6B6 Número de sector del último sector
C6B7 Acumulador al FDC
C6BA Longitud de GAP 3 en Read/Write
C6BC de disc-parameter-block al FDC
C6BF DTL, ha de ser hFF

* * * * Fase de ejecución de Read/Write/format

C6C1 Sector de I/O de 512 bytes
C6C4
C6C5 Leer estado del FDC, Drive Ready Write Prot?
C6CB
C6C9
C6CA Registro de estado 1 FDC
C6CD
C6CE => carry OK

C6CF
C6D0

* * * * Sector de Read/Write, 512 bytes

C6D1
C6D2 Acumulador al FDC
C6D5
C6D6 Buffer de I/O de 512 bytes
C6D9 Leer sector de OP-code?
C6DB en caso de que no, al bucle de escritura
C6DD al bucle de lectura

* * * * Bucle de lectura, leer hasta que FDC indique fin de sector
(bc) al registro de datos del FDC

C6E0 Leer byte de datos
C6E2 Almacenar (hl) en buffer
C6E3 (bc) al registro de estado del FDC
C6E4 Incrementar puntero de buffer
C6E5 C6C0 Incrementar byte de estado
C6E7 Esperar mensaje buyte-ready
C6EA Fin de ejecución, start result?
C6EC Recoger siguiente byte
C6EE

* * * * Bucle de escritura, escribir datos hasta que FDC indique fin

C6EF (bc) al registro de datos del FDC
C6F0 Recoger byte del buffer
C6F1 y escribirlo en disco
C6F3 (bc) al registro de estado del FDC
C6F4 Incrementar puntero de buffer
C6F5 Incrementar byte de estado
C6F7 Siguiente byte deseado?
C6FA Fin de ejecución, start result?
C6FC Escribir byte siguiente
C6FE

* * * * Busca la pista dada en d
C6FF Número de intentos de lectura

C702
C703 Buscar pista
C706 => pista hallada
C707 => 10 intentos sin éxito, READ FAIL
C709 Cantidad de intentos restantes
C70A
C70C Activar flag de recalibrar
C70E Pista deseada
C70F Pista 39
C711 Buscar pista
C714 Pista deseada
C715 Nuevo intento de hallar la pista

```
* * * * Fijar flag de recalibrado en recalibrate
C717
C718      Byte 17 en disc parameter actual
C71A      Determinar bloque, fijar flag de recalibrado
C71D      en recalibrate
C71F
C720

* * * * Dar salida a READ FAIL en búsqueda de pista
C722
C723
C724      Drive a c, enviar mensaje, CIR
C727
C728      Buscar pista en d, CALL (h1)
C72A

* * * *
C72B      Buscar pista en d, CALL (h1)
C72E
C72F
C730      Sense interrupt status del FDC
C733      Buscar pista en d, CALL (h1)
C736
C737
C738      Número de pista
C739      40 pistas
C73B
C73F
C740      Buscar pista en d
C743
C744      Buscar pista en d, CALL (h1)
C747
C748
C749      Número de pista
C74A      <>0?
C74B
C74E

* * * *
C74F
C750      Buscar pista en d
C753
C754      Buscar pista en d
C757
C758
C759      CALL (h1)
C75C
C75E
C75F      Buscar pista en d, CALL (h1)
C761
C762
```

*** ^87 buscar pista en el registro d
C763 Encender motor, sumar contadur para tiempo transcurrido
C766
C768
C769 Número de intentos de lectura
C76C
C76D Byte 17, flag de recalibrate
C76F del disc parameter block actual
C772 al acumulador
C773
C774 => ningün recalibrate
C776 b:= número de intentos de lectura
C777 registro de estado del FDC
C77A Recalibrate pista 0
C77C Dar salida al acumulador hacia el FDC
C77F Head select/unit select
C780 Dar salida al acumulador hacia el FDC
C783 Esperar 40 veces 12 milisegundos,
C785 luego leer estado de interrupciòn del FDC
C788
C78A Byte 16, DPB actual
C78C Èste es el nùmero de pista actual
C78F borrar
C791 Byte 17 en disc parameter block
C792 a -1
C794 b:= nùmero de intentos de lectura
C795
C796 Nùmero de pista alcanzado
C797 comparar con el nùmero de pista deseado
C798 => posicòn alcanzada
C79A b:= nùmero de intentos de lectura
C79B registro de estado del FDC
C79E Buscar pista de OP-code
C7A0 Dar salida al acumulador hacia el FDC
C7A3 Head select/unit select
C7A4 Dar salida al acumulador hacia el FDC
C7A7 Nùmero de pista deseada
C7AB Dar salida al acumulador hacia el FDC
C7AC Restarle nùmero de pista alcanzado
C7AE Posicòn alcanzada
C7B0
C7B1 Aguardar, luego leer estado de interrupciòn del FDC
C7B4
C7B5 Todo correcto, pista hallada
C7B7 Nuevo intento, eventualmente con recalibrate
C7B9 Contador de intentos desbordado?
C7BA En tal caso manejo de error
C7BD Sense interrupt status FDC
C7C0 Nuevo intento, eventualmente con recalibrate

* * * * La pista fue hallada
C7C2
C7C4
C7C5 Indicativo de todo OK
C7C6

* * * * Espera (acum. * 12)+16 ms lee estado de interrupción del FDC
C7C7
C7C8
C7CB Bucle de espera
C7CE
C7CF
C7D0 Bucle de espera finalizado?
C7D2
C7D5 Bucle de espera
C7D8 OP-code sense interrupt status
C7DA Dar salida al acumulador hacia el FDC
C7DD Leer estado de interrupción del FDC, drive READY?

* * * * Espera acum.*1 ms
C7E0 El acumulador es el índice del bucle
C7E1 Valor de demora
C7E3 Decrementar acumulador hasta llegar a cero
C7E4 Aprox. 1 milisegundo
C7E6 Decrementar índice del bucle
C7E7
C7E8 Eventualmente nuevo bucle
C7EA

* * * *

C7EB Buffer de HS/US
C7EE Número de drive hacia e
C7EF Valor h03 de disc parameter block, máscara de bloque
C7F1 cargarla en acumulador
C7F4 Incrementar máscara de bloque
C7F5
C7F8 y anotarla
C7F9 Transferir drive, pista y número de registro
C7FA de hBE53 hacia hBES4
C7FD
C7FF

* * * *

C800
C806
C807 Buffer de HS/US
C80A Tres bytes, drive, pista y número de registro
C80C (hl)=(de)?
C80D
C80E ==> distinto, error
C810
C811
C812 Siguiente byte

C814 Indicativo OK
C815

* * * *
C816
C81A

* * * * Espacio para este registro? En caso contrario siguiente pista
C81B
C820
C821 Número de drive hacia e
C822
C823 Número de registro en la pista
C824 Incrementar número de registro
C825 Byte bajo (low) de registros/pista
C826 de disc parameter block al acumulador
C829 Alcanzado número de registro máximo?
C82A => aún no alcanzado
C82C Poner a cero número de registro en la pista
C82E hl = BE5B, número de pista
C82F Incrementar número de pista
C830
C831

* * * *

C832
C838
C83B Reparar stack para caso de error
C83C => error
C83D Ningún error, bc de nuevo al stack
C83E Comprueba fin de sector
C841
C842 => record ya en buffer de sector
C844 Número de sector a c, número de buffer a hl
C847 Leer sector
C84A Manipular según eventual error
C84B hFF = ningún error en la lectura de sector
C84C
C850

* * * *
C851
C853

* * * *
C854 Flag de leer sector OK
C857
C858
C859 HS/US buffer
C85C
C85F Número de drive a e
C860 Cargar acumulador con antiguo número de drive

C861 Son ambos iguales?
C862 En caso negativo, RET
C863 Antiguo número de pista
C864 Nuevo número de pista
C865
C866 Son ambos iguales?
C867 En caso negativo, volver
C868 Desbordamiento al escribir registro?
C86B
C86C En caso de desbordamiento, volver
C86D Todo OK
C86E

* * * * *
C86F Borrar flag de leer sector OK
C872
C874
C875 (hl) => write sector flag
C876
C877
C878 => flag nulo, no escribir
C879
C87A Calcular número efectivo de sector
C87D ^BS escribir sector

* * * * *
C880
C883 HS/US buffer
C886
C887
C888 Valor de head select/unit select hacia e
C889
C88C
C88D Determina número de sector, comprueba overflow
C890
C891

* * * * * determina número de sector, comprueba overflow
C892 (número de registro)
C893
C894
C896 Cargar acumulador con número de registros/sector
C899
C89A Cargar acumulador con número de registro
C89B
C8A0

* * * * * calcular número efectivo de sector
C8A2 Números de drive y pista hacia e y d
C8A6 Cargar acumulador con disc parameter block h0F
C8A8 Primer número de sector de una pista
C8AB (número de sector deseado 0-8)

C8AE Da el número del sector a leer
C8AF Número del sector a c
C8B0 Determinar dirección buffer de sector
C8B3 hl=hl+iy

* * * * transferir registro a buffer de sector para escritura
C8B6 Rescatar todos los registros
C8B7
C8B9
C8BA Escribir indicativo de sector
C8BC Flag de Read/Write sector
C8BF Proporcionar bc, de y hl
C8C2 KL LDIR, transferir registro al sector
C8C5 Restaurar todos los registros.

* * * * transferir record del buffer de sector al de registro
C8C7 Rescatar todos los registros
C8C8
C8CA
C8CB Proporcionar bc, de y hl
C8CE
C8CF Transferir del buffer de sector al buffer de registro
C8D1 Restaurar todos los registros
C8D2
C8D5

* * * * de := comienzo de registro en buffer de sector
C8D6 HS/US buffer
C8D9 Head select/unit select a e
C8DA Cargar acumulador con valor h15 de DPB
C8DC Número de registros por sector
C8DF
C8E1
C8E2 (número de registro)
C8E3 Longitud de registro 128 bytes
C8E6 Offset al buffer de registro
C8E9 Corrección precisa
C8EA Calcula la dirección del siguiente
registro en el buffer de sector
C8EB
C8EC
C8EE Resultado a hl
C8EF de=de-iy, de=> siguiente registro
C8F2 Dirección de buffer de registro a hl
C8F5 Longitud de registro 128 bytes a bc
C8F8

* * * * Leer registro de estado del FDC, comprueba si DRIVE READY
C8F9 Leer fase de resultados
C8FC :=> sin aparición de errores
C8FD registro de estado 1 del FDC
C900 Disco protegido frente a escritura?
C902 => no protegido
C903 Mensaje de error 12, Disc is missing
C905

* * * * Leer registro de estado del FDC, comprueba si DRIVE READY
C807 Fase de resultados FDC, drive READY?
C80A :=> todo O.K.
C80B
C80C registro de estado 1 del FDC
C90F Disco protegido frente a escritura?
C911 => no protegido
C912 Mensaje de error 12, Disc is write prot.
C914 Drive a c, enviar mensaje, CIR
C917 'Ignore'
C918 'Cancel' => fin
C91B 'Retry'

* * * * Lleva al buffer los bytes de la fase de resultados
C81C
C81D
C91E Número de bytes leídos
C920 Dirección buffer fase de resultados
C923
C924 Registro de estado del FDC a bc
C926 Esperar a
C928 byte de estado ready
C92A Dirección del registro de datos del FDC
C92B Registro de estado de interrupción
C92D Dirección del registro de estado del FDC a bc
C92E Almacenar en (hl)
C92F
C930 Contador de número de bytes recogidos
C931
C933 Bucle corto
C934
C936 Leer byte de estado
C938 Bit de FDC busy
C93A Comando aún no finalizado
C93C
C93D
C93E Aislar código de interrupción, bits 6&7 de SRO
C940
C941 Anotar número de bytes de la fase de resultados
C942
C943

C944 Apareció error!
C945
C946 Final libre de errores

* * * * Sense interrupt status FDC

C947
C948 Registro de estado del FDC
C94B OP-code sense interrupt status
C94D Dar salida al acumulador hacia FDC
C950 Leer fase de resultados del FDC
C953 Mensaje 'INVALID COMMAND'?
C955 En caso contrario intentar de nuevo
C957
C958

* * * * Cargar acumulador con byte de DPB y darle salida
C959 Cargar acumulador con valor de DPB (acumulador)

* * * * Comprueba FDC, da salida al acumulador hacia FDC

C95C Valor a sacar dos veces al stack
C95D
C95E Bit 7, comprobar request for master
C960
C961 Esperar hasta que se reclame otro byte
C963 Bit 6, sentido de los datos al FDC
C964 Prueba de entrada o salida
C966 No enviar ningún byte al controlador,
C967 El FDC quiere enviar un byte al procesador
C968

* * * * El acumulador es transferido al FDC

C969 Valor a sacar de stack
C96A Registro de datos del FDC a bc
C96B Dar salida al byte hacia FDC
C96D Registro de estado del FDC a bc
C96E
C970 Bucle corto
C971 Decrementar acumulador de 5 a 0
C972
C974 Restaurar valor en el acumulador
C975

* * * * Encender motor, manipular stack, (hl) a buffer de I/O

C976 Memoria intermedia
C979 Recoger del stack dirección de RETURN
C97A Rescatar bc y de
C97B
C97C Anotar puntero de stack
C980 Dirección de RETURN de nuevo al stack
C981
C984 RETURN de nuevo a hl, C9AD al stack
C985 Dirección de RETURN definitiva al stack
C986 Rescatar de, bc y af, un RET a

C987 esta rutina lleva a C9AD!
C988 Un poco enrevesado, no es así?
C989 Del Ticker
C98C Motor Flag
C98F comprobar
C990 Motor en funcionamiento
C992 Dirección de port de control del motor
C995
C997 Encender motor
C999 Número de pasos
C99D Llamar al add Ticker, tiempo máximo funcionamiento motor
C9A0 Motor Flag
C9A3 comprobar
C9A4 Esperar a que flag de motor <> 0
C9A6 af, bc y de otra vez del stack
C9A7
C9A8
C9A9 (hl) apunta al buffer de I/O
C9AC El siguiente RET conduce a C9AD!

* * * *

C9AD Anotado aquí ante C97C
C9B1
C9B2 Número de pasos
C9B6 Llamar al add Ticker
C9B9
C9BA bc, de y hl fueron sometidos a PUSH ante C979
C9BB hacia C97B
C9BC
C9CA
C9CB (hl) => OBE4B
C9CC

* * * *

C9CD Dirección de Tick Block
C9D0 Reload count
C9D3 KL ADD TICKER

* * * * Rutina del contador de tiempo

C9D6 Motor Flag
C9D9 es 0 o bien hFF
C9DA deviene hFF o bien 0
C9DB Almacenar
C9DC en caso de Motor Flag nulo
C9DD => motor rodando
C9DF En caso contrario dirección de Tick Block
C9E2 KL DEL TICKER, desenclavar contador del motor

* * * *

C9E5
C9EB
C9EA Port de motor
C9ED Volver a fijar flip-flop de motor, apagar motor
C9EF
C9F0 Borrar Flag de motor
C9F3

* * * * Disponer disc parameter header y DP blocks

C9F4 Offset para DPH drive B
C9F7 Offset para DPB drive B
C9FA Inicializar DPH y DPB drive B
C9FD Offset para DPH drive A
CA00 Offset para DPB drive A
CA03 de=detiy, comienzo del DPB
CA06 Anotar dirección
CA0A y anotar en stack
CA0B h1=h1+iy, comienzo del DPH
CA0E Anotar dirección
CA11 y anotar en stack
CA12 Dirección del DPB en estándar
CA15 Transferir 25 bytes
CA18 a la dirección adecuada
CA1A Anotar en bc comienzo de la zona de suma de prueba
CA1B
CA1C Comienzo del DPH, éste es XLT
CA1D Eventual conversión del factor SKEW
CA1F No se utiliza, por tanto, 0

CA20
CA22

CA25 (h1)=> DIRBUF
CA26 Offset de los 128 bytes para el buffer de DIR
CA29 de=detiy, dirección del buffer de DIR
CA2C apuntarla en DIRBUF

CA2D
CA2F

CA30 Comienzo de los DPBs
CA31 apuntarlo en la cabecera

CA32
CA33

CA34 (h1)=> CSV, vector de suma de prueba
CA35 (bc)=> dirección del área de suma de prueba
CA36 apuntar en la cabecera (header)

CA37
CA38 (h1)=> ALV, Allocation vector
CA39 almacenar temporalmente en de
CA3A Offset entre área de suma de prueba y

```
CA3D      Allocation Area
CA3E      (h1)=> ALV
CA3F      apuntar área de asignación en el header
CA40
CA42

* * * * Standard DPB, Disc Parameter Block
CA43      STP registros por pista
CA45      BSH block-shift
CA46      BLM block mask
CA47      EXM extent mask
CA48      DSM número de bloques libres - 1
CA49      DRM número de entradas en DIR - 1
CA4B      ALO disposición del directorio
CA4D      CKS número de entradas a comprobar
CA4F      OFF offset de pista para pistas de sistema

* * * * Extensiones de los BPBs, no existentes en CP/M estándar
CA4B      Offset de registro para reconocimiento de formato
CA53      Número de sectores por pista
CA53      Longitud de GAP3 en Read/Write
CA53      Longitud de GAP3 en formateo
CA53      Byte de relleno en formateo
CA53      Bytes/sector para FDC, supone 512 bytes
CA53      Número de registros por sector
CA53      Tres memorias intermedias
CA53
CA53

* * * * Cargar acumulador con valor DPB (hAB90+(drive*h40)+acum.)
CA5C
CA5D
CA60      Acum.=(tabla+(drive*h40)+acum.)
CA61
CA62

* * * * h1 = pointer a DPB actual + acum.
CA63
CA64      Indicador a DPB drive 0
CA67      Head select/unit select - 1
CA68      Offset de los DPBs drive 0 y 1
CA6B      Salto si drive 0 actual
CA6D      Determinar comienzo tabla drive 1
CA6E      Acumulador apunta a byte deseado
CA6F      d=0, e=byte deseado
CA70
CA71

* * * * ^81 Mensajes de error del disc controller On/Off
CA72      Anotar antiguo valor en L
CA75      Anotar nuevo valor
CA78      Transferir antiguo valor al acumulador
CA79
```

* * * * Si enable dar salida a mensaje de error
CA7A Número de error al acumulador
CA7B Flag para mensajes de error
CA7E Comprobar
CA7F => no permitida la salida
CA81 Repetir número de error
CA82 Número de drive para salida a c
CA83 Dar salida a mensaje de sistema, C, I or R

* * * * Fin de salida no permitida
CA86
CA87 Borrar acumulador y flags
CA88

* * * * Hasta CA90 no utilizado
CA89
CA8F

* * * * BC = BC + IY
CA90
CA92 iy a hl, hl a stack
CA93 Sumar bc y hl
CA94 Resultado a bc
CA95
CA96 Restaurar hl
CA97

* * * * DE = DE + IY
CA98
CA9A iy a hl, hl a stack
CA9B Sumar de y hl
CA9C Resultado a de
CA9D Restaurar hl
CA9E

* * * * HL = HL + IY
CA9F
CAA0
CAA2 iy a de
CAA3 Sumar de y hl
CAA4 Restaurar hl
CAA5

* * * * Convertir minúsculas en mayúsculas
CAA6 'a' o mayor?
CAA8
CAA9 'z' o menor?
CAA8
CAA9 Convertir en mayúscula
CAA8

```
* * * * Borrar memoria desde (de) hasta (de)+(bc)
CAAF      Borrar acumulador
CABO      Borrar memoria
CAB1      Dirección siguiente
CAB2      Decrementar índice
CAB3      Comprobar si bc = 0
CAB4
CAB5      Borra (de) hasta (de+bc)
CAB7

* * * * Sacar error hacia A, luego comprobar 'IGN, RET, CHAN'
CADD      Sacar mensaje de sistema hacia a
CADD      Buscar y dar salida a
CABD      mensaje de sistema 20
CACO      KM READ CHAR
CAC3
CAC5      TXT CUR ON
CACB      KM WAIT CHAR
CACB      Convierte en mayúsculas
CACE      'C' = Chancel
CACD      => Z=1,C=0
CAC2      'I' = Ignore
CAC4      => Z=1,C=1
CAC5
CAC7      'R' = Retry
CAC9      => Z=0,C=0
CADC      Carácter 'BELL'
CADD      TXT OUTPUT, pitar una vez
CAFO      Esperar nueva entrada

* * * * Retry-entry, borrar flags
CAE2

* * * * Chancel- e Ignore-entry, dar salida a carácter en acumulador
CAE3      TXT OUTPUT
CAE6      TXT CUR OFF
CAE9      Dar salida a 'CR/LF'

* * * * SYSTEM-MESSAGE, buscar y dar salida a mensaje del sistema
CAEB
CAED
CAEE      Borrar bit 7 del número de error
CAFO      hl apunta a los mensajes de error y del sistema
CAF3      Número hacia b
CAF4      Incrementar en uno, vuelve a ser inmediatamente
CAF5      decrementado en el DJNZ
```

* * * * Releer los mensajes hasta el deseado
CAF7 Un carácter del mensaje al acumulador
CAF8 Incrementar indicador
CAF9 Comprueba fin de un mensaje
CAFA Si no lo es, seguir buscando
CAF0 b<>O releer mensaje siguiente

* * * * Dar salida a mensaje deseado
CAFE Un carácter del mensaje al acumulador
CAFF Incrementar indicador
CB00 Alcanzado fin del mensaje?
CB02 En caso afirmativo, salto
CB04
CB06
CB07 Seguir comprobando y dar salida a carácter en A
CB0A
CB0C
CB0D Recoger siguiente carácter

* * * * Dar salida a mensaje, listo
CB0F
CB12

* * * * Seguir comprobando y dar salida a carácter
CB13 Comprobar carácter en el acumulador
CB14 Si es menor que h80, entonces a salida
CB17 Cadena para el número de drive
CB19 Convertir número de drive en A/B y darle salida
CB1B Cadena para variable numérica
CB1D Determinar variable y darle salida
CB1F Cadena para el nombre de fichero
CB21 Dar salida a mensaje del sistema cadena de expansión
CB23 Nombre de fichero de 8 caracteres
CB25 Localizar en memoria y dar salida
CB28
CB2A darle salida
CB2D Extensión de 3 bytes
CB2F de apunta a la posición en memoria
CB30 Carácter al acumulador
CB31 Borrar bit 7
CB33 darle salida
CB36 Número - 1, siguiente carácter
CB38

* * * *
CB39 de apunta al nombre de fichero
CB3A "", carácter vacío
CB3C
CB48
CB49 Offset para carácter ASCII
CB4B darle salida

* * * *

CB4D

CB56

* * * *

CB58

CB59 Cero en el acumulador?
CB5A en tal caso, dar salida a un espacio
CB5C Offset para carácter ASCII
CB5E
CB5F darle salida

* * * *

CB61 Número de drive 0 ó 1
CB62 Sumar 'A'
CB64 Dar salida a 'A' o 'B'

* * * * Dar salida a carácter a posición de cursor en ventana

CB66 Almacenar carácter en memoria intermedia

CB67 Carácter vacío?

CB69 En caso afirmativo, darle directamente salida

CB6B

CB6C

CB6D TXT GET WINDOW

CB70 TXT GET CURSOR

CB73 d contiene la columna derecha de la ventana

CB74

CB77

CB79 h contiene la columna del cursor

CB7A

CB7E

CB7F Dar salida a 'CR/LF'

* * * *

CB82 Carácter de nuevo al acumulador

CB83 TXT OUTPUT

* * * * Mensajes de error / mensajes del sistema

* * * * Mensaje 0 del sistema

CB96 CR/LF

* * * * Mensaje 1 del sistema

CB89 dar salida a tres espacios

* * * * Mensaje 2 del sistema

CB8D 'num. Variable'K

* * * * Mensaje 3 del sistema

CB90 'CR/LF"CR/LF'"num. Variable'K free

* * * * Mensaje 4 del sistema
CB99 'CR/LF'Bad command'CR/LF'

* * * * Mensaje 5 del sistema
CBA7 'CR/LF Filename' already exists

* * * * Mensaje 6 del sistema
CBB8 'CR/LF Filename' not found

* * * * Mensaje 7 del sistema
CBC4 'CR/LF Drive A/B' directory 'full'

* * * * Mensaje 8 del sistema
CBD1 'CR/LF Drive A/B Disc' 'full CR/LF'

* * * * Mensaje 9 del sistema
CBD4 'CR/LF Drive A/B Disc' changed,
CBDC closing 'Filename' 'CR/LF'

* * * * Mensaje A del sistema
CBE9 'CR/LF Filename' is 'read' only

* * * * Mensaje B del sistema
CBF5 'Filename'

* * * * Mensaje C del sistema
CBF7 'CR/LF Drive A/B' user 'num. Variable'

* * * * Mensaje D del sistema
CBFF ...^C

* * * * Mensaje E del sistema
CC05 'CR/LF failed to load' CP/M 'CR/LF'

* * * * Mensaje F del sistema
CC0C 'CR/LF failed to load' boot sector 'CR/LF'

* * * * Mensaje 10 del sistema
CC1A 'CR/LF Drive A/B' 'read' 'fail CR/LF'

* * * * Mensaje 11 del sistema
CC1E 'CR/LF Drive A/B' 'write' 'fail CR/LF'

* * * * Mensaje 12 del sistema
CC22 'CR/LF Drive A/B disc is 'write'
CC2A protected 'CR/LF'

* * * * Mensaje 13 del sistema
CC33 'CR/LF Drive A/B disc 'missing' 'CR/LF'

* * * * Mensaje 14 del sistema
CC3D 'CR/LF' Retry, Ignore or Chancel?

* * * * Mensaje 15 del sistema
CC58 'CR/LF' Drive 'A/B':

* * * * Mensaje 16 del sistema
CC63 'CR/LF' Failed to load

* * * * Mensaje 17 del sistema
CC74 'CR/LF' 'CR/LF'

* * * * Mensaje 18 del sistema
CC77 'CR/LF' Drive A/B' disc

* * * * Mensaje 19 del sistema
CC7E Fail 'CR/LF'

* * * * Mensaje 1A del sistema
CC85 Full 'CR/LF'

* * * * Mensaje 1B del sistema
CC8B 'CR/LF' 'Filename'

* * * * Mensaje 1C del sistema
CC8F write

* * * * Mensaje 1D del sistema
CC95 read

* * * * No utilizado
CC9A FF RST 38H
CC9F FF RST 38H

* * * * Parchear todos los vectores de cassette para disco
CCA0
CCA1 Drive y usuario a default A0
CCA4
CCA7 Acum. = hFF
CCAB flag de OPENIN activo a hFF inactivo
CCAB flag de OPENOUT activo a hFF< inactivo
CCAE
CCB2 Salvaguardar
CCB5 los vectores de cassette
CCB8 de=de+iy, mempool + h164
CCBB 13*3 bytes = 13 vectores
CCBE
CCC0
CCC1 hOCD30 es la dirección de entrada para
CCC3 todas las entradas de cassette parcheadas
CCC4
CCC6
CCC7 KL ASC Curr SELECTION, número de la

CCCC ROM de floppy como tercer byte para RST
CCCB Código para return .

CCCD
CCDO

* * * * DISC
CCD1 Disc out
CCD4 Apareció error, entonces RET

* * * * DISC IN
CCD5 Cass in open
CCD8 Parchear cass in open y las
CCDA siguientes seis Entries
CCDD
CCDE Catalog
CCE1
CCE2 Parchear catalog

* * * * DISC OUT
CCE4 Cass out open y las
CCE7 siguientes cuatro Entries

* * * *
CCE9 Siguen parámetros?
CCEA En caso afirmativo, salto
CCEC En caso contrario, parchear vectores deseados
CCEF de=d+iy, discmem + h18B
CCF2 restart 3
CCF4
CCF5 Todas las Entries indican hacia hA88B
CCF6
CCFC

* * * * TAPE, restaurar vectores de cassette
CCFD Restaurar tape out
CD00 Apareció error, entonces RET

* * * * TAPE IN
CDO1
CDO4
CDO7 7 vectores, tape in todos
CDOA
CDOD Apareció error
CDOE
CD11
CD14 Un vector cass catalog
CD16

* * * * TAPE OUT
CD18
CD1B
CD1E 5 vectores, todos Tape Out
CD21 Siguen paràmetros?
CD22 En tal caso dar salida a error
CD24 hl=hl+iy
CD27
CD29 Activar carry como indicativo de todo OK
CD2A

* * * *
CD2B Mensaje del sistema 4, 'BAD COMMAND'
CD2D buscarlo y darle salida

* * * * Se accede desde todas las CAS-Entries mediante RST3
CD30 Inicio de memoria para disco a iy
CD34 necesario para la utilizaciòn de los
CD35 juegos alternativos de registros
CD36
CD37 El contenido de c es variable
CD38 Direcciòn de retorno de salto a de
CD39 POP en dos RET m s
CD3A
CD3B Este RET ha de provocar un avance
CD3C bc y RET original de nuevo al stack
CD3D
CD3E Restaurar c y b
CD3F
CD41 Aumentar direcciòn de TRET en h10D2
CD44 y al stack como nueva direcciòn
CD45 RET apuntar  a la tabla siguiente
CD46 Repetir juego de registros original
CD47
CD48 Ahora pueden volver los INTs
CD49 Alli hay un RET!

* * * * Bloque de salto para las CAS/DISC-Entries parcheadas
CD4C DISC IN OPEN
CD4F DISC IN CLOSE
CD52 DISC IN ABANDON
CD55 DISC IN CHAR
CD58 DISC IN DIRECT
CD5B DISC RETURN
CD5E DISC TEST EOF

CD61 DISC OUT OPEN
CD64 DISC OUT CLOSE
CD67 DISC OUT ABANDON
CD6A DISC OUT CHAR
CD6D DISC OUT DIRECT

CD70 DISC CATALOG

* * * * Aumentar puntero de stack y a discmem+6

CD73 Preciso para la

CD76 corrección del stack

* * * * Puntero de stack a discmem+6

CD77

CD78 Dos CALL y el PUSH HL = 6 bytes

CD7B Stack correspondientemente corregido a hl

CD7C y anotar en discmem+6

CD7F y en discmem+7

CD82

CD83

* * * * Almacenar stack, si OPENIN no activo entonces ruptura

CD84 Almacenar puntero de stack

CD87

CD88 Flag de OPENIN activo

CD8B

* * * * Almacenar stack, si OPENOUT no activo entonces ruptura

CD8D Almacenar stack

CD90

CD91 Flag de OPENOUT activo

CD94 Fichero no abierto?

CD96 => ruptura

CD98 Si es necesario login, proporcionar format

CD9B

CD9C

* * * * Ruptura si OPENIN activo

CD9D Flag de OPENIN activo

CDA0

* * * * Ruptura si OPENOUT activo

CDA2 Flag de OPENOUT activo

CDA5 Almacenar stack

CDA8 Flag vale hFF si no activo

CDA9 => no activo, todo OK

CDAA Número de error al acumulador

CDAC Borrar flag de carry, indicativo de error

CDAD Restaurar stack, interrumpir comando

* * * * Dar salida a bad command, interrumpir comando

CDAF Mensaje del sistema 4, 'BAD COMMAND'

CDB1

CDB4 Número de error al acumulador

CDB6 Hasta ahora no se dió salida a ningún mensaje de error

CDB8 Borrar carry, indicativo de error

* * * * Interrumpir comando
CDB9 Restaurar puntero de stack
CDCB
CDBF
CDC0 y retorno

* * * * Comprueba si acumulador = 2, si no interrup. y 'bad command'
CDC1

* * * * Comprueba si acumulador = 1, si no interrup. y 'bad command'
CDC2
CDC3 => acumulador nulo
CDC4 Dar salida a bad command, interrumpir comando

* * * * Recoge longitud cadena a b, direcciòn cadena a hl
CDC7 Recoger un paràmetro y a hl
CDCA Longitud de la cadena
CDCB (hl) => direcciòn de la cadena
CDCC LD HL,(HL)

* * * * Recoger en hl un paràmetro de la ampliaciòn de órdenes
CDCF Byte bajo y
CDD2 Byte alto del paràmetro de transmisiòn a hl
CDD5 ix al eventual paràmetro siguiente
CDD7
CDD9

* * * * !A:
CDDA Acumulador a 0, valor para drive A
CDBB

* * * * !B:
CDDD Acumulador a 1, valor para drive B
CDDF Almacenar puntero de stack
CDE2 Transmitir valor al DOS

* * * * !DRIVE
CDE4 Almacenar stack
CDE7 Un paràmetro de la serie, en caso contrario 'bad command'
CDEA Recoger paràmetro
CDED
CDEE Dar salida a bad command, interrumpir comando
CDF1 Indicativo de drive deseado (A/B) al acumulador
CDF2 Convierte en mayùsculas
CDF5 Da 0 ò 1
CDF7 Login
CDFA Transferir nùmero de user al DOS
CDFD

* * * * !USER
CDFE Asegurar stack
CE01 Un paràmetro de la secuencia, en caso contrario 'Bad command'
CE04 Recoger paràmetro a hl

CE07 Número màximo de user + 1
CEOA de = hl?, nùmero de user legal?
CEOD demasiado alto, 'Bad command', ruptura
CE10 Transferir nùmero de user al DOS
CD13

* * * *

CE14 Primer carácter expand. filename, nùmero de drive
CE15
CE19

CE1A Número de drive de drive comunicado
CE1B hFF = OPEN con drive comunicado activo
CE1D Flag de OPENIN activo
CE20 en este drive?
CE21 => OPENIN activo
CE23 Flag de OPENOUT activo
CE26 en este drive?
CE27 => OPENOUT activo
CE29 h00 = ningùn OPEN activo en drive comunicado
CE2B
CE2C

CE2D Comprueba nùmero de drive, proporciona disc-format
CE30
CE31

CE32 Comprueba si hl es 0000
CE33 Si hl es 0000, entonces drive no READY
CE34 Dar salida a bad command, interrumpir coando
(hl)=> disc-parameter-header (A910/A920)
CE37 a iy+3/iy+4
CE3D flag acerca de si OPEN activo en drive comunicado
CE40 Nùmero de drive (HS/US)
CE43
CE47

* * * * Copia nombre expandido de fichero en OPENIN-header-block
CE48 Offset hacia OPENIN-header-block
CE4B
CE56

* * * * Copia nombre expandido de fichero en OPENOUT-header-block
CE57 Offset hacia OPENOUT-header-block
CESA
CE5B

CE5C hl=hl+iy, hl => header-block
CE5F
CE61

CE62 (de) => dirección del buffer de usuario
CE63

CE64 a puntero de comienzo del buffer de usuario
CE65

CE66 Vector en buffer de usuario
CE67 inicializarlo
CE68

```

CE69      (hl)=> comienzo nombre de fichero en cabecera
CE6A
CE6B      (bc)=> EFN a partir de número de usuario
CE6C      Número de bytes a borrar
CE6F
CE70      Borra desde (de) hasta (de+bc), resto de la cabecera
CE73      (bc)=> EFN a partir de número de usuario
CE74      Transferir a hl
CE75
CE76      Dirección en cabecera del nombre de fichero a de
CE77
CE78      Longitud nombre de fichero incluyendo drive/user
CE7B      transferir al header-block
CE7D      Dirección en cabecera nombre de fichero a hl
CE7E      Dirección buffer de usuario a de
CE7F
CE83
CE84      Indicativo tipo de fichero 'unprot. ASCII'
CE86
CE88
CE89      Dirección buffer de usuario a de
CE8A
CE91

* * * * Suma comprobación de dos bytes extendida a la cabecera (h43 bytes)
CE92      hl=> comienzo de la cabecera
CE93      Poner suma de comprobación a 0
CE96      Byte high de de a 0
CE97      Número de bytes
CE99      hl = indicador en cabecera, suma de comprobación al stack
CE9A      un carácter al acumulador
CE9B      Siguiente carácter en la cabecera
CE9C      hl = suma de comprobación, indicador en 1 cabecera al stack
CE9D      Acumulador a e, de'= carácter en 16 bits
CE9E      Incorporar a suma de comprobación
CE9F      Otro carácter? =>
CEA1      Suma de comprobación a de
CEA2      hl => comienzo cabecera
CEA3

* * * *
CEA4
CEAC

* * * * DISC IN OPEN
CEAF      Anotar puntero de stack
CEB2      Dirección de buffer de 2K
CEB3      Comprobar validez de nombre de fichero
CEB6      Disc-parameter-header a hl, en caso contrario login
CEB9
CEBC      (bc)=> nombre expandido de fichero
CEBD      (hl)=> primer carácter extensión
CEBE      Primer carácter extensión = hFF?

```

CEBF => nombre de fichero dado sin extensin
 CEC1 Buscar nombre de fichero en el directorio
 CEC4 => fichero no hallado, ruptura
 CEC7

 * * * * Nombre de fichero dado sin extensin
 CEC9 Apuntar 3 espacios como extensin
 CECC Nombre de fichero sin extensin en disco?
 CECF => hallado
 CED1 Apuntar 'BAS' como extensin
 CED4 Almacenado como fichero BASIC?
 CED7 => hallado
 CED9 Apuntar 'BIN' como extensin
 CEDC Almacenado como fichero binario?
 CEDF
 CEE0 Fichero no hallado, 3 espacios como extensin -
 CEE3
 CEE4 => ruptura, 'File not found'

 * * * * Nombre de fichero hallado en el directorio
 CEE7 Direccin buffer de 2K
 CEE8 Copiar nombre de fichero en cabecera de OPENIN
 CEEB Direccin de comienzo de cabecera
 CEEC
 CEEF de=de+iy, direccin OPENIN-FCB
 CEF2
 CEF3 Nmero de drive
 CEF4 a OPENIN-FCB
 CEF5 Nmero de caracteres en fichero de input a 0
 CEF8
 CEFB h1=h1+iy, direccin de record-buffer
 CFEF Record a Record-Buffer
 CF01 => aparcio error
 CF03 (h1)=> comienzo de Record-Buffer
 CF04 (de)=> nombre de fichero en OPENIN-FCB
 CF05 Suma de comprobacin de h43 bytes del registro a de
 CF08 ld h1,(h1), suma de comprobacin almacenada, en su caso
 CF0B h1 = de? en caso afirmativo, el registro es cabecera
 CF0E
 CF0F
 CF10 => suma de comprobacin <>, fichero ASCII!
 CF12
 CF15 de=de+iy, OPENIN-header-block+5
 CF18 Nmero de bytes de cabecera
 CF1B transferir al OPENIN-header-block
 CF1D

 * * * * Ningn fichero ASCII como fichero de input
 CF1F Nmero de caracteres en fichero de input a 0
 CF22
 CF27
 CF28 Direccin, en la que se escribi originalmente el fichero, a de
 CF29

CF2A
CF2C
CF2D Longitud del fichero a bc
CF2E
CF30
CF31 Indicativo OPENIN OK
CF32 Ningún error al sistema operativo CPC
CF33 Tipo de fichero del fichero abierto
CF36

* * * * DISC OUT OPEN

CF37 Anotar puntero de stack
CF3A Dirección de buffer de OPENOUT de 2K
CF3B Comprueba validez nombre fichero, si válido dispone EFN
CF3E Disc parameter header a h1, en caso contrario login
CF41 Buffer de OPENOUT de 2K
CF42 Copiar EFN en OPENOUT-header-block
CF45 (h1) => OPENOUT-header +5, número de usuario
CF46 Apuntar extensión '\$\$\$' en DHB
CF49 Fichero ya en el disco?
CF4C Dirección de OPENOUT-header a h1
CF4D
CF4E (h1) => número de drive
CF4F
CF52 de=de+iy, OPENOUT-filecontrol-block
CF55 Longitud del nombre de fichero con drive y user
transferir a OPENOUT-FCB
CF58 Longitud disposición bloques en el dir(h10)+puntero(7)
CF5D Borra en FCB el resto tras nombre de fichero
CF60 Header-block +5
CF61 Indicativo OPENOUT OK
CF62 OPENOUT OK al sistema operativo CPC
CF63

* * * * DISC IN CHAR

CF64 Rescatar todos los registros
CF65
CF66
CF67 Recoger un carácter del buffer de OPENIN
CF6A Restaurar registros
CF6B
CF6C
CF6D Indicativo de aparición de error
CF6E Leido indicativo EOF?
CF70 Indicativo todo OK
CF71 => no EOF
CF72 En caso contrario borrar carry
y retorno
CF73

* * * * Recoge un carácter de fichero de OPENIN abierto
CF74 Salvaguardar stack, comprobar flag de OPENIN
CF77 Disc-mempool
CF79 hacia de
CF7A
CF7D (hl)=> Indicativo in char(1)/in direkt(2)
CF7E Indicativo al acumulador
CF7F Disc in direkt activo hasta ahora?
CF81 En tal caso error, interrupción del comando
CF84 Apuntar indicativo disc in char
CF86 Comprobar si hay carácter en el buffer
CF89
CF8A Contador de tres bytes de caracteres de fichero
CF8B
CF8E
CF8F => ningún carácter en el buffer
CF91
CF98
CF99 Rellenar buffer de OPENIN de 2K
CF9C Número de caracteres leídos en buffer de OPENIN
CF9D Comprobar a nulidad
CF9E
CF9F => error, no hay caracteres en el buffer
CFA1 Número de bytes leídos a bc
CFA2
CFA3
CFA4 Es leído un carácter, decrementar por tanto
CFA5 Número restante en el buffer
CFA6 Anotar carácter
CFA7
CFA8
CFBB Puntero en buffer de OPENIN a de
CFBC
CFBE
CFBF LD A,(HL), recoger carácter del buffer
CFC0
CFC1 Incrementar puntero de buffer de OPENIN
CFC2
CFC3
CFC4
CFC5 Indicativo de un carácter recogido
CFC6

* * * * Fin de error
CFE7 Código de error
CFC9 Borrar carry
CFCA

* * * *

CFCB
CFD1
CFD2 ld hl,(hl), dirección de buffer
CFD5 anotar en stack
CFD6 16 registros
CFD9 a cargar en buffer de OPENIN, si existen
CFDC
CFDE Determinar número de registros leídos
CFDF Número hacia b
CFE0
CFE2 Determinar número de bytes leídos
CFE4 Resultado hacia bc
CFE6 Puntero de buffer de OPENIN
CFE7 (hl)=> número de bytes leídos
CFEB anotarlo
CFE9
CFEE
CFEF Anotar puntero de buffer de OPENIN
CFF0
CFF4

* * * * DISC IN DIRECT

CFF5 Salvaguardar stack, comprobar flag de OPENIN
CFFB Dirección de carga
CFF9
CFFC hl=hl+iy
CFFF Comprobar indicativo in char(1)/in direct
D000 En caso de dis in char,
D002 error, interrupción del comando
D005 Apuntar indicativo disc in direct
D007
D00A
D00B Número de caracteres hacia de
D00C
D00F
D010 Dirección de carga a hl
D011 e intercambiar
D012 2^7-128
D014 Divide número de caracteres/128
D017 Resultado, número de registros a bc
D018
D019 Dirección de carga hacia hl
D01A Cargar número calculado de registros
D01D
D01E => error
D020
D027
D02A hl=hl+iy
D02D
D045
D046 ld hl,(hl)

* * * *

D049

* * * * Registros en buffer de OPENIN, nùmero de registros en dc
 D04B Registro, eventualmente de disco, en buffer
 D04E => fin de fichero o cualquier otro error
 D04F
 D052 de=de+iy, dirección en cabecera de OPENIN de tipo de fichero
 D055 Comprobar tipo de fichero
 D056
 D057 => carry únicamente en 'Protected File'
 D05A Longitud de registro
 D05D Aumentar puntero de buffer
 D05E Número de registros restantes a leer
 D05F Comprobar número a nullidad
 D060
 D061 => todavía no todos leídos
 D063 Todo OK, todos los registros leídos
 D064

* * * * DISC TEST EOF
 D065 Llamada a disc in char
 D068 RET ante EOF, en caso contrario devolver carácter al buffer

* * * * DISC RETURN

D069
 D06C
 D06F h1=h1+iy
 D072
 D08E

* * * * DISC OUT CHAR
 D08F Anotar stack, comprobar flag de OPENOUT activo

D092
 D094
 D095 El acumulador contiene el carácter
 D096 Disc-mempool a de
 D098
 D099
 D09C Disc-out-mode flag (char/direkt)
 D09D de la cabecera al acumulador
 D09E Disc-out-direkt hasta ahora?
 DOA0 => cierre del comando
 DOA3 Apuntar indicativo disc-out-char
 DOA5
 DOA8 (h1):= nùmero de caracteres en el buffer de OPENOUT
 DOA9 1d h1,(h1), nùmero de caracteres a h1
 DOAD
 DOB1
 DOB2 => más de 2K, escribir datos en disco
 DOB5
 DOB6

DOB7 Incrementar caracteres en el buffer del usuario
DOB8 Valor de 2 bytes
DOB9 Incrementar byte alto
DOBB sólo en caso de necesidad
DOBC
DOBF
DOC0 Aumentar File Character Counter
DOC3
DOC6 (hl) => vector a puntero en buffer de usuario
DOC7
DOC8 bc => puntero en buffer de usuario
DOC9
DOCB
DOCC Depositar carácter de acumulador en buffer de usuario
DOCD Incrementar puntero en buffer de usuario
DOCE También byte alto en caso de necesidad
DODO
DOD4
DOD5 Indicativo todo OK
DOD6 Mensaje al sistema operativo CPC todo OK
DOD7

* * * * DISC OUT DIRECT

DOD8 Anotar stack, comprobar flag de OPENOUT activo
DODB Acumulador = tipo de fichero
DODC Dirección a partir de la que ha de escribirse
DODD Longitud de los bloques de datos
DODE
DOE1 hl=hl+iy, disc-out-mode (char/direkt)
DOE4 Al acumulador
DOE5 Disc-out-char activo hasta ahora?
DOE7 En tal caso error, cierre del comando
DOEA Apuntar indicativo de direkt
DOEC
DOEF
DOFO bc = dirección de entry
DOF1
DOF2
DOF3 bc = longitud bloque de datos
DOF4
D111
D112 Tipo de fichero al acumulador
D113
D116
D117 Tipo de fichero a OPENOUT-header-block

* * * * Escribir en disco 2K (buffer de usuario con OUT CHAR)

D118 Disc mempool
D11A a de
D11B
D11E
D11F Byte de cabecera 'First lock'

D120
D121 => no es el primer bloque
D123
D126 (hl):= tipo de fichero
D127 Tipo de fichero al acumulador
D128 El nibble high es irrelevante
D12A Indicativo 'unprotected ASCII'?
D12C => es el caso
D12E
D131 hl=> OPENOUT-file-control-block
D132 Disc-mempool
D133
D134 Número de registros +1 para Header-Record
D137 Comprobar número de registros en el FCB
D13A
D13F
D140 Block Char Counter
D141 Número de caracteres a de
D142
D143
D146 hl=> vector de buffer de usuario
D147 ld hl,(hl), hl => buffer de usuario
D14A
D14B Buffer a registros, luego a disco
D14E bc => buffer de usuario
D14F hl => character count
D150 ponerlo a cero
D152
D160
D161 Indicativo OK
D162
D163

* * * *

D164 Número de caracteres en el bloque
D165 Número de registros/bloque
D167 Número a de
D168 Divide número caracteres/128
D16B Resultado, registros precisos a de
y bc
D16D
D16E Transferir registros
D171 Número de caracteres
D172 Byte low al acumulador
D173 Limitar a un registro
D175 Si es nulo, no más bytes
D176 En caso contrario, resto de fichero en nuevo registro
D177 : luego byte high = cero
D179
D17C de=de+iy
D17F Buffer de record
D180 KL DIR, transfiere último registro al buffer
D183 Indicativo EOF

D185 Encadenarlo
D186 hl=> buffer de registro
D187 bc = aumentar Record-Counter
D188

* * * * Escribir registros en fichero (número en bc)

D18A
D18B
D18E de=de+iy
D191 Tipo de fichero al acumulador
D192 Bit 0 activado?
D193 => no activado, no 'Protected'
D195
D196
D199 de=de+iy
D19C de = buffer de registro
D19D Longitud de registro
D1A0 KL DIR, Header-Block a buffer de registro
D1A3
D1A4
D1A5 'proteger' registro
D1AB Registro en buffer de sector, eventualmente en disco
D1AB hl=> buffer de record
D1AC Longitud de record
D1AF
D1B0 Rebajar número de registros
D1B1 Escritos todos los registros?
D1B2
D1B3 => registros aún por escribir
D1B5

* * * * DISC IN CLOSE

D1B6 Salvaguardar stack, comprobar flag de OPENIN
D1B9 Apagar motor, 'desenclavar' event

* * * * DISC IN ABANDON

D1BC Fijar flag de OPENIN en inactivo
D1CO Cierre

* * * * DISC OUT ABANDON

D1C2 Salvaguardar stack, comprobar flag de OPENOUT
D1C5
D1CB de=de+iy
D1CC Liberar de nuevo bloques en tabla de asignación
D1CF
D1D2
D1D3 Buscar pista 0
D1D6 Cierre

* * * * DISC OUT CLOSE
D1D8 File Character Count
D1DB hl=hl+iy
D1DE Comprueba si de hecho fueron transferidos
D1DF caracteres,
D1E0
D1E2
D1E3 en caso contrario, disc out abandon, ning n apunte de dir
D1E5 Salvaguardar puntero de stack, comprobar flag de OPENOUT
D1E8 Transferir  ltimo registro al buffer
D1EB
D1EE de=detiy, nombre de fichero en el OPENOUT-FCB
D1F1 .
D1F2 Apuntar nombre de fichero y disposici n en dir de bloques
D1F5
D1F8 bc=bc+iy,bc:= OPENOUT-header-block
D1FB
D1FE (hl):= tipo de fichero
D1FF
D203
D204 Primer car cter de extensi n al acumulador
D205 Comprobar si es hFF
D206 => extensi n indicada
D208 Tipo de fichero al acumulador
D209
D20B
D20D Apuntar extensi n 'BAS'
D210

* * * *
D212 Tipo de fichero 2?
D214
D216 Apuntar extensi n 'BIN'
D219

* * * *
D21B Apuntar 3 espacios de extensi n
D21E Direcci n OPENOUT-header-block a hl
D21F
D220 Tipo de fichero al acumulador
D221 El nibble high es irrelevante
D223 'unprotected ASCII'?
D225 => est  'protected'
D228 OPENOUT-FCB a bc
D229
D22B Indicativo OPENOUT no activo
D22C
D22D Sobreescribe '\$\$\$' con extensi n original
D230 Indicativo DISC OUT CLOSE OK
D231 Mensaje al sistema operativo CPC todo OK
D232

* * * *

D233
D236 H ning n OPEN activo en drive referido
D23A
D240
D243 de=de+iy
D246
D24F

* * * *

D252
D25A

* * * * 'Protected File', protecci n mediante XOR
D25C Puntero de buffer de OPENIN a hl
D25D RAM LAM, car cter de buffer al acumulador
D25E
D263
D264 Byte de nuevo a buffer de OPENIN
D265 Incrementar puntero de buffer
D266 Y de nuevo al stack
D267 Siguiente XOR-byte
D269 Siguiente XOR-byte
D26A Contador de bytes para tabla-ix
D26B => tabla a n no finalizada
D26D Tabla de 11 bytes para ix
D26F Inicio de tabla a ix
D273 Contador de bytes para tabla-hl
D274 => tabla a n no finalizada
D276 Tabla de 13 bytes para hl
D278 Inicio de tabla
D27B b:= n mero de bytes a codificar
D27D
D280

* * * * Tabla para ix
D281
D287

* * * * Tabla para hl
D28C
D298

* * * * Extensiones
D299
D2A5

* * * *

D2A8 tres espacios
D2A9

```
*****  
D2AB      '$$$' para nombre de fichero intermedio  
D2AD  
  
*****  
D2AF      'BAK' para fichero back-up  
D2B1  
  
*****  
D2B3      'BAS' para ficheros BASIC  
D2B5  
  
*****  
D2B7      'BIN' para ficheros binarios  
D2B9  
D2BA      Sumar byte low del inicio de tabla  
D2BC      Resultado a e  
D2BD      Sumar a byte high de carry  
D2BF      Restar byte low  
D2C0      Byte alto resultado a d  
D2C1  
  
***** Apuntar extensin en nombre de fichero  
D2C3  
D2C4  
D2C7      de=de+iy  
D2CA  
D2CB  
D2CC      Longitud nombre de fichero incluyendo user, sin extensin  
D2CF  
D2D0      Longitud extensin  
D2D3  
D2D4      Apuntar extensin deseada en nombre de fichero  
D2D6  
D2D9  
  
*****  
D2DA      Longitud nombre de fichero + extensin  
D2DD  
D2E2  
D2E4      Nmero de apuntes de dir & tabla de asignacin a 0  
D2E7  
D2EB  
D2EC      Siguiente apunte de dir a (de)  
D2EF  
D2F0      => ningn apunte ms en el DIR  
D2F2      Encadenar extensin 'BAK'  
D2F5      Busca en el disco el nombre de fichero dado  
D2F8      => BAK no existente  
D2FA  
D2FC      Prueba si el fichero es de READ ONLY
```

```

D2FF      => es de READ ONLY
D301
D302      Apuntar extensi n en (de)
D305      Busca en el disco el nombre de fichero dado
D308      => fichero no existente
D30A
D30C      Prueba si el fichero es de READ ONLY
D30F      => el fichero es de READ ONLY
D311
D327
D329      N mero de apuntes de dir & tabla de asignaci n a 0
D32C
D333

* * * *
D335      Apuntar extensi n 'BAK'
D338      Busca en el disco el nombre de fichero dado
D33B      => hallado
D33E      Buscar en dir fichero-'$$$'
D341      => hallado
D342      Encadenar extensi n original
D345      Busca en el disco el nombre de fichero dado
D348      => no hallado
D349
D34B
D34C      Apuntar extensi n 'BAK'
D34F

* * * * Prueba si el nombre de fichero con '$$$' se halla en el disco
D351      Apuntar extensi n '$$$'
D354      Busca en el dir el nombre de fichero dado
D357      => no hallado
D358      (bc)=> nombre de fichero
D359      Direcci n del apunte de dir id ntico
D35A      a bc
D35B      Escribir extensi n original en registro de dir
D35E
D35F

* * * *
D362      N mero de apuntes de dir & tabla de asignaci n a 0
D365      Proporcionar siguiente apunte de dir, de = n mero
D368      => ning n apunte de dir m s
D369      Busca nombre de fichero temporal ('$$$') y original
D36C      Proseguir b squeda

* * * *
D36E      N mero de apuntes de dir & tabla de asignaci n a 0
D371      Proporcionar siguiente apunte de dir, de = n mero
D374      => ning n apunte de dir m s
D375      Si hallado fichero temporal ('$$$'), extensi n orig. a registro
D378      En caso contrario, proseguir b squeda

* * * *

```

D37A

D37D

D37E Apuntar extensin en (de)

D381 Busca en el disco el nombre de fichero dado

D384

D387

* * * * READ-ONLY-file, interrumpir comando

D388 Apuntar extensin en (de)

D38B

D38C

D38D Mensaje de sistema 10, file is read only

D38F darle salida, comando finalizado

* * * *

D392

D396

D399 de=de+iy, direccin de OPENIN-FCB

D39C Comprobar si se ley  ltimo registro

D39F => ning n registro m s en el fichero

D3A1 de:= n mero de registro, hl nombre de fichero en en FCB

D3A2 (hl):= buffer de registro

D3A3 Recoger registro en buffer de registro

D3A6

D3A7

* * * *

D3A9

D3AE

* * * * Registros a buffer de registro, en caso contrario a disco

D3AF Buffer de registro

D3B0

D3B1 N mero de registros

D3B2 y de nuevo buffer de registro

D3B3

D3B6 de=de+iy, OPENOUT-FCB

D3B9 Prueba si DISK FULL

D3BC => a n queda espacio

D3BE Mensaje de sistema 8, disc full

D3C0 darle salida, interrumpir comando

D3C3 Busca apunte de dir libre

D3C6

D3D1

D3D2 Buscar y ocupar bloque libre

D3D5

D3D6 Dar salida a mensaje de sistema 8, disc full e

D3D8 interrumpir comando, si ning n bloque libre

D3DB Anotar bloque ocupado

D3DC

D3E9
D3EA Registro a buffer de registro, en caso contrario a disco
D3ED
D3EE
D3F1 Número de registros +1
D3F4
D3F6
D3F7 Indicativo todo OK
D3F8

* * * *

D3F9
D40B
D40A Registro a buffer de registro, en caso contrario a disco
D40D Dar salida a bad command, interrumpir comando

* * * * Comprueba si se leyó último registro
D410
D422
D423 32 bytes de (hl) a (de)
D426
D42D

* * * * !DIR
D42E Salvaguardar stack
D431
D433 Siguen parámetros?
D434 => no más parámetros
D436 Uno de los parámetros es OK en caso contrario 'Bad command'
D439 b:= longitud cadena, hl:= dirección cadena
D43C Convertir en nombre de fichero correcto
D43F Disk parameter header a hl, en caso contrario login
D442 Dar salida a 'Drive #: user #'
D445 Longitud de un nombre de fichero, '.' inclusive
D447 Determinar cantidad de apuntes/linea de pantalla
D44A
D44B
D44C Número de apuntes de dir & tabla de asignación a 0
D44F Buscar nombre de fichero y proporcionar disposición
D452 => ningún apunte más
D454 Comprueba si el apunte lleva atributo de SYS
D457 => segundo carácter extensión >7F, atributo de SYS
D459 Apuntes contados al stack
D460
D461 h:= apuntes/linea
D462 l:= apuntes de salida pendiente en linea
D463 => dar salida a tres espacios
D464 => dar salida 'CR/LF'
D465 Dar salida a nombre de fichero, de apunta a nombre fichero
D466 Apuntes mostrados en linea actual
D467 => linea aún no llena
D468 Linea llena, número a valor inicial
D469

D46B Apuntes contados a hl
D46C Apunte siguiente

* * * * Cierre !DIR

D46E
D46F Determinar y dar salida a número de bloques libres

* * * * Determinar número de apuntes/línea para IDIR y CAT
D472 Longitud del apunte más los tres espacios

D474

D476

D477 TXT GET WINDOW

D47A Columna derecha de la ventana actual

D47B

D47D

D47F Inicializar contador de apuntes/línea

D481 Número de apuntes/línea

D482 Longitud de un apunte de DIR

D483

D485 Número de apuntes de DIR/linea, corrección

D486 Si número = 0, no formatear,

D487 fijar cifra en 1

D489

* * * * ERA

D48A Salvaguardar stack

D48D Un parámetro de los que siguen, si no, 'Bad command'

D490 Dirección nombre del fichero a borrar a hl

D493 Poner nombre de fichero para DOS

D496 Disk parameter header a hl, en caso contrario, login

D499 Número de apuntes de dir & tabla de asignación a 0

D49C Buscar nombre de fichero y proporcionar disposición

D49F => comunicar fichero no hallado, interrupción

D4A1

D4A4 Buscar nombre de fichero y proporcionar disposición

D4A7

D4A9

* * * *

D4AA

D4B0

* * * *

D4B1 Comprueba si el fichero es de READ ONLY

D4B4

D4B5 Mensaje de sistema 'Filename is read only'

D4B7 => dar salida a mensaje, interrupción

D4BA

D4BB Liberar bloques en tabla de asignación

D4BE Indicativo fichero borrado

D4C0 apuntarlo en nombre de fichero

D4C1

*** * * * REN**

D4C4 Salvaguardar stack
D4C7 Dos paràmetros de los que siguen, si no, interrupciòn
D4CA Primer paràmetro, nuevo nombre, a h1
D4CD Dispone nombre de fichero para DOS
D4D0
D4D1 Segundo paràmetro, nombre antiguo, a h1
D4D4 Disponer nombre de fichero para DOS
D4D7
D4D9
D4DA Dar salida a bad command, interrumpir comando
D4DD Disk parameter header a h1, en caso contrario, login
D4E0
D4E1
D4E2 Comprobar si el nuevo nombre de fichero ya existe
D4E3
D4E8
D4E9 Número de apuntes de dir & tabla de asignaciòn a 0
D4EC Buscar antiguo nombre de fichero y proporcionar disposiciòn
=> fichero no hallado
D4EF
D4F1 Comprueba si el fichero es de READ ONLY
=> el fichero es de READ ONLY, no modificar
D4F7
D4F9
D4FA longitud de nuevo nombre de fichero
D4FD sobreescribir nombre antiguo
D4FF
D502
D505 Buscar antiguo nombre de fichero y proporcionar disposiciòn
D508
D50B

*** * * * Fichero no hallado, interrupciòn**

D50C Direcciòn nombre de fichero para salida a de
D50D
D50E Mensaje de sistema 6, file not found
D510 darle salida, interrumpir comando

*** * * * CATALOG**

D513 Anotar puntero de stack
D516 Direcciòn de buffer de usuario
D517 transferir a ix
D519 Longitud del buffer de usuario
D51C Borra buffer de usuario (de) hasta (de+bc)
D51F
D522 Disk parameter header a h1, en caso contrario, login
D525 Salida de 'Drive #: user #'
D528
D529
D52A Número de apuntes de dir & tabla de asignaciòn a 0
D52D Buscar antiguo nombre de fichero y proporcionar disposiciòn
D530
D532 Comprueba si el fichero lleva atributo de SYS

D535 => es fichero de SYS, no darle salida
D537 Un apunte en buffer de usuario
D53B
D53C Siguiente apunte, por lo que aún queda otro
D53E Longitud de un apunte de dir en monitor
D540 Determina número de apuntes/línea
D543 Número a d
D544
D54C
D54E Número de líneas
D54F Buffer de usuario
D551 a hl
D552 Lineas a c
D553 Apuntes/linea a b
D554
D555 Salida de un apunte
D558
D55A
D55C hl:= hl * 14 para salida alfabética
en varias columnas
D560
D561 y dar salida a siguiente apunte
D563
D56F
D571 Proporciona número de bloques ocupados
D574
D576 Mensaje de sistema 3, xxxK free
D577 darle salida

* * * * Salida de un apunte desde el buffer de usuario
D57A RAM LAM, LD A,(HL) desde buffer de usuario
D57B Carácter un 0?
D57C Entonces ningún apunte y =>
D57D rescatar registros
D57E
D57F
D580 Apuntes/linea
D581 igual A apuntes sacados?
D582 En caso de desigualdad, dar salida a 3 espacios
D585 En caso contrario dar salida a 'CR/LF'
D588
D589 Dar salida a nombre de fichero
D58C Comprueba si el fichero es de READ ONLY
D58F '*'
D591 Si R/O, dar salida a '*'
D593 'espacio'
D595 TXT OUTPUT
D598
D59B
D59C RAM LAM, LD A,(HL) de RAM
D59D
D59E

D59F RAM LAM, LD A,(HL) de RAM
D5A0
D5A1 Mensaje de sistema 2, tres espacios
darle salida
D5A3
D5A6
D5A9

* * * *

D5AA
D5AC
D5AE Buffer de usuario
D5B0 a hl
D5B1 RAM LAM, LD A,(HL) de RAM
D5B2 Comprueba si carácter 0
D5B3
D5C8

* * * *

D5CA
D5CB Determina número de bloques ocupados del fichero
D5CE
D5D2
D5D3 RAM LAM, LD A,(HL) de RAM
D5D4
D5D5
D5D6 RAM LAM, LD A,(HL) de RAM
D5D7
D5E1

* * * *

D5E3
D5E4
D5E6
D604

* * * *

D605
D607
D608 Indicativo de fin en el buffer de usuario
D60A
D60B de:= puntero de registro
D60C
D60D Longitud nombre de fichero+extensión
D610 KL LDIR, de registro a buffer de usuario
D613 Puntero de registro a hl
D614 Inicio de registro a hl
D615 Inicio de registro a de
D616 Determina número de bloques ocupados del fichero
D619 Número de bloques a de

D61A
D61B Apuntar en el buffer de usuario
D61C
D622

* * * *

D623
D62D
D62E RAM LAM, LD A, (HL) de RAM
D62F
D639

* * * * hl:=hl*14
D63A Anotar
D63B
D63C
D63D hl=antiguo valor en hl *2
D63E hl=antiguo valor en hl *3
D63F hl=antiguo valor en hl *6
D640 hl=antiguo valor en hl *7
D641 hl=antiguo valor en hl *14
D642
D643

* * * *

D644 Número de apuntes de dir & tabla de asignación a 0
D647 Buscar nombre de fichero y proporcionar disposición
D64A
D64C Mensaje de sistema 5, file already exists
D64E darle salida, interrumpir comando

* * * * Buscar nombre de fichero en el directorio
D651 Número de entradas de dir & tabla de asignación a 0
D654 Buscar nombre de fichero y proporcionar disposición
D657 => no hallado
D659 Número de entrada de dir hallado
D65A
D65D hl=hl+iy, dirección OPENIN-FCB
D660 a de
D661 32 bytes, entrada de dir en OPENIN-FCB
D664 Número de entrada de dir hallado
D665 Fichero activo en este drive?
D668
D669
D66A => fichero activo
D66B Leer directorio hasta fin, proporcionar número de ficheros
D66E y número de bloques ocupados
D670 Directorio leido
D671 Marcar fichero como activo en este drive?
D675

* * * * Busca nombre fichero en directorio, proporciona disposición h
D676 Número de entradas de dir & tabla de asignación a 0
D679 Buscar nombre de fichero y proporcionar disposición
D67C => no hallado
D67E
D681

* * * * Determina disposición de bloques en disco
D683 => nombre de fichero
D684 Buscar pista 0
D687
D688
D68B Fichero activo en este drive?
D68E
D68F => fichero abierto
D690
D691 Tabla de asignación a 0, ocupar bloques de dir
D694
D695

* * * * Buscar nombre fichero, disposición bloques y número ficheros
D698 Número de bloques en tabla de asignación, contar ficheros
D69B => ningún apunte más
D69C Nombre de fichero dado=apunte de dir?
D69F => distintos, buscar y proporcionar disposición
D6A1 Hallado nombre de fichero idéntico

* * * * Proporcionar número de ficheros en disco
D6A2 Contador de entradas
D6A3 OPEN activo en este drive
D6A6
D6A7 => OPEN activo
D6A9 Record a buffer, puntero de record=> apunte de dir
D6AC => ningún apunte más
D6AD Indicador de entrada de dir
D6AE Indicativo de fichero borrado
D6B0
D6B1 => si apunte borrado
D6B2 En caso contrario aumentar número de entradas
D6B5
D6B7 Determina número de bloques ocupados

* * * * Llegada de salto D6A2 si OPEN activo
D6BA Comprueba si la posición de la entrada de dir es OK
D6BD
D6BE

* * * * Número de caracteres en fichero a h1
D6C1
D6C4
D6C5 ld h1,(h1)

```
* * * *
D6C8
D6E8
D6EA      Máscara de exent al acumulador
D6ED
D6F9

* * * *
D6FA
D704
D707      Borra (de) hasta (de+bc)
D70A
D70B

* * * *
D70C
D720
D721
D728
D729      h1 = de? carry si son iguales
D72C
D72E

* * * *
D72F
D732
D734      Máscara de bloque al acumulador
D737
D739
D73B      Block-shift al acumulador
D73E
D741
D743      Número máximo de bloques, byte high al acumulador
D746
D757
D758      ld h1,(h1)
D75B

* * * *
D75D
D76D
D76F      Block-shift al acumulador
D772
D77A

* * * *
D77B
D77C

* * * * Determinar número de registros ocupados en el FCB
D77D
D78B
```

* * * * Ocupa apunte de directorio libre

D78C
D78D
D78E Buscar apunte libre
D791 Dirección de apunte al stack, hl=dirección nombre fichero
D792 No se precisa número de drive
D793 Escribir nombre fichero y disposición de bloques en registro
D796
D797 Escribir nombre fichero y disposición bloques en registro dir
D79A
D79B

* * * * Poner a 0 número caracteres en fichero

D79C
D7A6

* * * * Incrementar en 1 número caracteres en fichero

D7A7
D7B2

* * * *

D7B3 Número de entradas de dir & tabla de asignación a 0
D7B6 Buscar nombre de fichero y proporcionar disposición
D7B9

* * * * Buscar apunte de directorio libre

D7BB
D7BE
D7BF Comprueba espacio en el directorio
D7C2 mensaje del sistema 7, dar salida a directory full
D7C4 Interrumpir comando si no hay espacio
D7C7 (de)= puntero de registro en registro de dir
D7C8 Apunte borrado?
D7CA => no libre, siguiente apunte (entrada)
D7CC
D7CD OPEN activo en este drive?
D7D0
D7D1
D7D3 Error, interrumpir comando
D7D6
D7D7

* * * * Busca en el directorio el nombre de fichero dado en bc

D7D8 (bc) => nombre de fichero del fichero deseado
D7D9 (de) => puntero de record

D7DA
D7DB Dirección del nombre de fichero dado a hl

D7DC

D7DD Primer carácter entrada dir, número usuario

D7DE igual a número de usuario dado?

D7DF En caso de que no =>

D7E1

D7E2

D7E3 Longitud nombre de fichero + extensiòn
D7E5
D7E6 Comodín '?' en el nombre de fichero dado?
D7E8 => sobreleer caràcter en la entrada dir
D7EA Caràcter de entrada dir
D7EB comparar con caràcter en el nombre dado
D7EC
D7EE => los nombres son distintos
D7F0 En caso contrario caràcter siguiente nombre dado
D7F1 Caràcter siguiente entrada dir
D7F2 comprobarlo
D7F4
D7FB
D7FA Màscara de exent al acumulador
D7FD
D810
D811 Fichero no hallado
D812 Indicativo OK
D813

* * * * Borra tabla de asignaciòn, apunta bloques de dir
D814
D816 Número màximo de bloques a h1
D819
D81B Divide por 8 el nùmero de bloques
D81E Correcciòn, 22 bytes para tabla de asignaciòn
D81F
D820
D822 (h1) => comienzo de tabla de asignaciòn actual
D825 8 bloques = indicar 1 byte como libre
D827 Siguiente byte
D828 Decrementar nùmero
D829 Borrados los 22 bytes?
D82A
D82B => no todos borrados aun
D82D
D82F Magnitud del directorio dada en bloques a h1
D832
D833
D835 (h1) => comienzo de tabla de asignaciòn actual
D838 Apuntar bloques de dir como dispuestos
D839
D83B

* * * * Dispone bloques de este apunte en tabla de asignaciòn
D83C Número de apuntes de dir comprobados
D83D Indicador al comienzo del apunte en registro
D83E
D83F
D840 Offset de la disposiciòn de bloques
D843
D844 Número de los apuntes de disposiciòn/apunte de dir
D846

```

D848
D84A    Número màximo de bloques, byte high al acumulador
D84D    Si es 0 apunta de 1 byte
D84E    => apunta de 1 byte
D850
D854
D855    No mès bloques dispuestos
D857
D858
D85A    Número màximo de bloques a h1
D85D
D85E    Número de bloque válido?
D85F
D860
D861    Válido=> disponer/liberar en tabla de asignaciòn
D864
D86B

* * * * Disponer/liberar nùmero de bloques en tabla de asignaciòn
D86C
D86D    Número de bloques dispuestos
D86E
D870
D872    Dividir número de bloques por 8
D875    Resultado a de, byte a tabla de asignaciòn
D876
D878    (h1) => comienzo de tabla de asignaciòn actual
D87B    Saltar número preciso de bytes
D87C    Número de bloques dispuestos a de
D87D
D883
D884    Determinar posiciòn de bit en tabla de asignaciòn actual
D885
D88D
D88E    Aplicar OR a la muestra de bits resultante
D88F    y almacenar en tabla de asignaciòn
D890

* * * * Busca bloque libre en tabla de asignaciòn
D893
D895
D897    Número màximo de bloques a h1
D89A
D89B
D89D    (h1) => comienzo de tabla de asignaciòn actual
D8A0    b:= contador de bits, c:= muestra de bits
D8A3    Apunte de tabla de asignaciòn al acumulador
D8A4    Muestra de bits para bloque
D8A5    => hallado bloque libre
D8A7    Comprobar bit siguiente
D8A8
D8A9    de:= bits por comprobar
D8AA    Ningùn bloque mès?

```

```

DBAB      => no se halló bloque libre alguno, disc full
DBAD      Decrementar número de bits por comprobar
DBAE      Bucle extendido a 8 bits
DBB0      Byte siguiente a tabla de asignación
DBB1      Seguir comprobando

* * * * Da núm. de bloque a partir posición de bit, lo pone en tabla as
DBB3      Muestra de bits de tabla de asignación al acumulador
DBB4      Ocupar bloque libre hallado
DBB5      y retirar de tabla de asignación
DBB6
DBB8      Número máximo de bloques a hl
DBBB      .
DBBC      Calcular número de bloque, a hl
DBBE      Indicativo de hallado bloque libre
DBBF
DBC1

* * * * Número bloques de tabla asign. ocupados para indicación de DIR
DBC2
DBC3
DBC4      hl es contador de bloques
DBC7      Al stack
DBC8
DBC9      Número máximo de bloques a hl
DBC0      y a de
DBC1
DBC0      (hl)=> comienzo ALV actual
DBC3      b:= contador de bits, c:= muestra de bits
DBC6      Byte de tabla de asignación al acumulador
DBC7      Posición de bit indicada como ocupada?
DBC8      => no ocupada
DBC9      Contador a hl
DBCDB     Incrementar contador
DBCDC     y de nuevo al stack
DBCDD     Muestra de bits al acumulador
DBCDE    Recoger por rotación siguiente muestra de bits
DBCDF    y anotar de nuevo en c
DBE0      de:= número de bits por comprobar
DBE1      Número ya nulo?
DBE2      => ningún bloque más por comprobar
DBE4      Decrementar número de bits por comprobar
DBE5      Bucle de bits
DBE7      Byte siguiente de tabla de asignación
DBE8      Seguir comprobando

```

* * * * Proporciona el número efectivo de bloques ocupados
D8EA Recoger del stack contador de bloques ocupados
D8EB Proporciona disposición efectiva a partir de BSH y hl
D8EE Resultado a de
D8EF
D8F1

* * * * Determina el número de bloques ocupados del fichero
D8F2 Comienzo de una entrada de dir en registro de dir
D8F3 Offset de entradas de disposición de bloques
D8F6
D8F7 d:= bytes tabla asignación, e:= contador bloques ocupados
D8FA
D8FC Número máximo de bloques, byte high al acumulador
D8FF Si es 0, valor en sólo 1 byte a tabla de asignación
D900 Un byte de tabla de asignación
D901 Incrementar puntero en tabla de asignación
D902 Byte high número máximo bloques =0, entonces sólo un byte
D904 En caso contrario comprobar byte low
D905
D906
D907 Si 0, comprobar tabla de asignación
D908 Ningún bloque ocupado
D90A Incrementar contador bloques ocupados
D90B Decrementar número
D90C Quedan bytes en la tabla?
D90E Número a hl
D90F
D910
D912 Block-shift al acumulador
D915
D91A

* * * *

D91C
D91D bc=> nombre de fichero temporal
D91E Número de apuntes en registro comprobados
D91F 4 posibles, 0-3
D921
D923 Número de apuntes a de
D924
D926 Máximo de apuntes de DIR a hl
D929 hl = de? comprobados todos los apuntes posibles?
D92C
D92D
D92E => leídos todos
D930 Nuevo registro a buffer de registro
D933

D935

D937 => buffer de registro, de nuevo al principio
D93A Longitud de un apunte en el buffer (32 bytes)

D93D

D93E

* * * *

D940 Puntero de registro a siguiente inicio de dir

D941

D943 Puntero de registro a de

D944

D947

* * * *

D948

D94A Número de apuntes comprobados/4

D94D Resultado, número de registros comprobados a de

D94E

D950 (h1) => número de registro

D953 Recoge registro, número en h1, en buffer de registro

D956

D958 Número de apuntes de DIR a comprobar a h1

D95B

D95C h1 = de? comprobados todos los apuntes posibles?

D95F

D960 => leídos todos los apuntes

D961

D963 (h1) => bloque de suma de comprobación

D966

D967 Calcular suma de comprobación extendida al registro

D96A Es igual a la suma de comprobación almacenada?

D96B => son iguales

D96C

D974

D975 Error

D978

D979

* * * *

D97A Número del apunte de dir dispuesto

D97B Dirección de buffer

D97C

D97E Divide h1/4

D981 Resultado, número de registros a h1

D982

D984 (h1) => buffer de registro

D987

D989 Registro de dir a buffer de registro, en caso contrario a di

D98C

D98E Número de registros de dir a comprobar a h1

D991

```
D992      hl = de? comprobados todos los registros posibles?  
D995  
D998  
D99A      (hl) => buffer de suma de comprobacin  
D99D  
D99E      Suma de comprobacin extendida al registro  
D9A1      y apuntar en buffer de suma de comprobacin  
D9A2  
D9A3  
D9A4      Comprueba la correcta posicin del nombre de fichero  
D9A7      Error  
D9A8  
D9A9  
D9AB      Incrementar nmero de apuntes dispuestos  
D9AC  
D9AE      (hl) => nmero de apuntes de directorio  
D9B1      Anotar nuevo nmero  
D9B2  
D9B7
```

```
* * * * Comprueba que la posicin del apunte de directorio es OK  
D9B8  
D9B9      Nmero de apuntes de directorio  
D9BA  
D9BC      (hl) => nmero de apuntes de dir dispuestos  
D9BF      Nmero a de  
D9C0  
D9C2  
D9C3      hl = de, apunte calculado OK?  
D9C6  
D9C7
```

```
* * * * Suma de comprobacin extendida al registro, resultado al acumulador  
D9C8  
D9C9  
D9CA      Longitud de registro  
D9CC  
D9CE      (hl) => buffer de registro  
D9D1      Borrar acumulador  
D9D2      suma de comprobacin extendida al registro  
D9D3  
D9D8
```

```
* * * * Comprobar atributo de READ ONLY, bit 7 carcter 1 extensin  
D9D9  
D9DA      Primer carcter extensin  
D9DD
```

```
* * * * Comprobar atributo de SYS, bit 7 carcter 2 extensin  
D9DF  
D9EO      Segundo carcter extensin  
D9E3  
D9E4      Carcter al acumulador
```

D9E5 Carry activado con atributo activado
D9E6
D9E7

* * * * Recoger registro, número en de, en buffer de registro
D9E8
D9EA
D9EB Determinar pista y sector a partir de número de registro
D9EE Leer registro en buffer de registro
D9F1 Mensaje de error en el acumulador

* * * * Escribir registro, número en de, en buffer de sector
D9F3
D9F6
D9F7 Determinar pista y sector a partir de número de registro
D9FA
D9FB Escribir registro en buffer de sector
D9FE Mensaje de error en el acumulador
D9FF => error, interrumpir comando
DA02
DA05

* * * * Calcula pista y sector a partir de número de registro
DA06 Número de registro
DA07 Dirección de buffer para un registro
DA08 a bc y
DA09 transferir a rutinas del controlador
DA0C
DA0D
DA0F Offset de pista a hl
DA12 y bc
DA13
DA14
DA15 Número de registros/pista a hl
DA18 Corrección
DA19
DA23
DA24 Pista en c a BE54 para el controlador
DA27
DA28
DA29 (hl) => (A910/A920)
DA2C
DA2D Convertir número de registro
DA30
DA31
DA32 Sector a BE55 para el controlador

```
* * * * Carga hl con disc parameter header + acumulador
DA35      Suma byte low de DPH actual
DA38      Resultado a l
DA39      Suma byte high, eventualmente con carry
DA3C
DA3D      High byte a h
DA3E

* * * * Carga hl con contenido (disc parameter header + acumulador)
DA3F      (HL)=>DPH+acumulador
DA42      ld hl,(hl)

* * * * Carga hl con contenido (DPB+acumulador)
DA45
DA46
DA48      (hl) => inicio disc parameter block
DA4B      Offset deseado
DA4C
DA4D
DA4E      Calcula la dirección precisa
DA4F
DA50
DA51      LD HL,(HL)
* * * * Carga acumulador con contenido (disc parám. block + acum.)
DA54
DA55      Carga hl con contenido (DPB + acumulador)
DA58      Byte deseado al acumulador
DA59
DA5A

* * * *
DA5B
DA5E

* * * *
DA60
DA6B

* * * *
DA6A      Dispone EFN, comprueba nombre de fichero
DA6D

* * * * Dispone expanded filename, comprueba espacios y '?'
DA6F
DA71
DA74      Disponer el nombre de fichero en el buffer
DA77
DA7B      Longitud del nombre de fichero extensión incluida
DA7A
DA7B      bc => nombre de fichero
DA7C      Primer carácter del nombre al acumulador
DA7D      '?' comodín
DA7F      En caso afirmativo=> dar salida a bád command
```

DAB1
DAB2 Comprobar si carácter siguiente es '?'
DAB4 Ningún interrogante hallado
DAB5 Parece OK

* * * *

DAB6
DAB8

* * * *

DABD Espacio
DABF Dirección de buffer de registro
DA92 Disponer fichero
DA95
DA9F

* * * * Dispone nombre expandido de fichero, comprueba si es OK
DAA0 Dispone EFN
DAA3 => espacio en nombre de fichero, bad command
DAA5

* * * *

DAA6 Espacio
DAA8 Offset de Record-Buffer
DAAB
DAB5

* * * *

DAB6
DAB7 de=de+iy, buffer para nombre expandido de fichero
DABA Anotar
DABB Número de drive activo
DABE apuntarlo en nombre de fichero
DABF
DAC0 Número de user activo
DAC3 apuntarlo en nombre de fichero
DAC4
DAC6
DAC7 El nombre de fichero tiene 8 caracteres de longitud
DAC9 8 espacios a (de) hasta (de+7)
DACC
DACD La extensión tiene 3 caracteres de longitud
DACF 3 veces hFF a (de+8) hasta (de+hOA)

DAD2
DAD5 Borra de (de+hOB) hasta (de+hOD)
DAD8 b => longitud de nombre de fichero de input, c => h0FF
DAD9 de => inicio de expanded filename (EFN)
DADA hl => dirección de input filename (IFN)
DADB
DADC Dispone EFN si IFN es correcto
DADF
DAE0 Bad command, filename de input no correcto
DAE2 Dirección EFN a bc

```

DAE3
DAE4      de => nombre de fichero
DAE5
DAE6      Primer carácter del IFN
DAE7      Espacio?
DAE9      Dar salida a bad command, interrumpir comando

* * * * Comprueba IFN, dispone EFN con DRIVE y USER
DAEA      Dummy

* * * *
DAED      Comprobar longitud IFN
DAEE
DAF1
DAF2      Apareció error! Ningún nombre de fichero?
DAF3
DAF5
DAF6      El carácter es ':'?
DAF8      Sí, entonces =>
DAFA      Un carácter de IFN, convert. uppercase
DAFD      Comprobar si ':'
DAFF      Ningún ':' hallado en el nombre

* * * * Si ':", comprobar validez del número de USER
DB00
DB02
DB03      Salto, si ningún ':' hallado
DB05
DB06      '0'
DB08      Carácter menor que '0', por lo que no se trata de una cifra
DB0A
DB0C      Carácter mayor o igual a ':", por lo que no es una cifra
DB0E      Es una cifra, restándole h30 se obtiene en binario
DB10
DB11      Apuntar cifra en EFN
DB12      Un carácter de IFN, convert. uppercase
DB15      '0'
DB17      Carácter menor que '0', por lo que no se trata de una cifra
DB19      ':'
DB1B      Carácter mayor o igual a ':", por lo que no es una cifra
DB1D      El carácter es una cifra
DB1E      Era 1 la primera cifra?
DB1F      No, error en el IFN, número de USER erróneo
DB20      Comprobar si la segunda cifra vale de 0 a 6 (ambos inclus.)
DB22
DB24      Segunda cifra >6, número de USER erróneo
DB25      Apuntar cifra en EFN
DB26      Un carácter de IFN, convert. uppercase

* * * * Si no es cifra comprobar validez (lógica) de número de drive
DB29
DB2A      'Q'
DB2C      El carácter es 'Q', 'R' ... error en el IFN

```

DB2E 'A'
 DB30 Caràcter menor que 'A', error en el IFN
 DB32 Restar 'A', el resultado da de 0 a 15
 DB34 Estos son los drives (lògicamente) posibles
 DB35 Un caràcter de IFN, convert. uppercase

 * * * * En caso de USER y/o drive, ha de seguir ':', si no, error
 DB38 Recoger caràcter, sobreleer espacio
 DB3B Ahora ha de venir un ':'
 DB3D En caso contrario, error en el IFN
 DB3E Recoger caràcter, sobreleer espacio
 DB41
 DB42 Error, no hay ningùn caràcter tras ':'

 * * * * De IFN determina nombre de fichero y extensiòn en cuestiòn
 DB43
 DB44
 DB45 ':'
 DB47 Falta nombre de fichero, sólo dada la extensiòn
 DB48 Longitud del EFN
 DB4A
 DB4D
 DB4E Ahora ha de venir un '.'
 DB50 Error, IFN excesivamente largo
 DB51 Recoger caràcter, sobreleer espacio
 DB54 Longitud de la extensiòn
 DB56 Ningùn caràcter mès, entonces extensiòn = 3 espacios

 * * * * Comprueba caràcter para filename y extensiòn, apunta en EFN
 DB58 ','
 DB5A Si no hay espacios rellenar con espacios
 DB5C
 DB5D
 DB5E Almacenar temporalmente
 DB5F Tabla de caracteres 'prohibidos' en el nombre
 DB62 Recoger valor de tabla
 DB63 Incrementar puntero
 DB64
 DB65 Fin de la tabla?
 DB67 Comparar valor de tabla con caracter
 DB68 En caso de desigualdad, siguiente valor
 DB6A Indicativo de valor 'prohibido'
 DB6B Caràcter al acumulador
 DB6C
 DB6D
 DB6E Caràcter 'prohibido', resto con espacios
 DB70 Contador longitud nombre o extensiòn
 DB71 Todos los caracteres por? luego =>
 DB72 '*' comodin para el resto
 DB74 Si comodin, completar el resto con ' '?
 DB77 Caràcter OK, apuntar en nombre de fichero expandido
 DB79 Un caràcter de nombre de fichero, convert. uppercase
 DB7C Ningùn caràcter mès, rellenar resto con espacios

DB7E Caràcter de espacio?
DB80 En caso negativo, comprobar y apuntar

* * * * Tirar resto en IFN tras espacio
DB82 Comprueba si es espacio, si lo es, siguiente caràcter

* * * * Rellena memoria (de) hasta (de+c) con espacios
DB85
DB86 Valor para caràcter de espacio
DB88
DB8D

* * * * Rellena memoria (de) hasta (de+c) con '?'
DB8E '?', comodin para un caràcter

* * * * Rellena memoria (de) hasta (de+c) con caràcter en acumulador
DB90 Corraccion precisa
DB91 Siguiente posiciòn de memoria
DB92 o eventualmente c=0, en tal caso fin
DB93 Caràcter a memoria direccionada
DB94 Incrementar puntero
DB95 Y de nuevo

* * * * Recoge un caràcter, comprueba si long. filename >0
DB97 Un caràcter, comprueba longitud de nombre de fichero
DB9A La longitud es 0, error!

* * * * Prueba si espacio en acum., si lo hay, recoge proximo caràcter
DB9B Espacio?
DB9D
DB9E Era otro caràcter
DB9F Recoger caràcter siguiente
DBA2 Comprueba si es espacio, si lo es, siguiente caràcter
DBA4

* * * * Recoge caràcter de nombre fichero, lo convierte en mayùscula
DBA5 Longitud (restante) de nombre de fichero
DBA6 Longitud <> 0?
DBA7 Si es 0, RETURN
DBA8 (hl) => puntero de IFN
DBA9
DBAA RAM LAM. LD A,(HL) de RAM
DBAB Unicamente caracteres ASCII, por favor
DBAC Convierte en mayùsculas
DBBD Indicativo byte recogido
DBB1

* * * * Caracteres prohibidos en el nombre de fichero
DBB2 '<'
DBB3 '>'
DBB4 '.'
DBB5 ','
DBB6 ';'
DBB7 ':'
DBB8 '='
DBB8 Abrir corchete
DBB8 Cerrar corchete
DBB8 Underline
DBB8 '%'
DBB8 Corchete de SHIFT
DBBE '('
DBBE ')'
DBBE '/'
DBBE Backslash
DBBE Delete
DBBE Indicativo fin de tabla

* * * * Da salida a tres espacios, número de drive actual a c
DBC4 Mensaje de sistema 1
DBC6

* * * * Da salida a 'filename', número de drive actual a c
DBC8 Mensaje de sistema 11
DBCA
DBC9 Número de drive a c
DBCE

* * * * Da salida a 'Drive #: user #', e=Drive, c=núm. user
DBD0
DBD1 Número de drive
DBD2 a e
DBD3
DBD5
DBD6 Número de user
DBD7 a c
DBD8 Mensaje de sistema 12
DBDA Da salida a mensaje de sistema
DBDD
DBDE

* * * * Transfiere con LDIR 32 bytes de hl a de
DBDF
DBEA

* * * * Divide hl por (acumulador**2)
DBEB
DBF2

```
* * * * Compara h1 con de
DDF3
DHFB

* * * * Carga h1 con (h1)
DBF9
DBFF
* * * * dc00 a dfff no se utilizan
DC00
```


Capítulo 5: Programas y trucos para la DDI-1

5.1. Errores en MERGE y CHAIN MERGE

Este capítulo es interesante para todo aquel que compró su floppy en los primeros tiempos. Se coló lamentablemente un error en el sistema operativo, de modo que al volver a cargar programas con las órdenes MERGE y CHAIN MERGE aparece injustificadamente el mensaje de error EOF met.

Dado que la firma Amstrad sabe de la existencia de este error, en versiones posteriores del sistema operativo ya no aparecerá; es el caso del CPC 664 y del CPC 6128. Aquellos de entre ustedes que no sepan en cual de los dos casos se halla su floppy, pueden dilucidarlo en este capítulo.

Como ya mencionamos, y quizás hayan comprobado en su propia carne algunos de ustedes, el AMSDOS contiene un ligero fallo en el sistema operativo. No desista por ello; los sistemas más satisfechos y caros todavía tienen innumerables fallos en el sistema operativo.

Y que no cunda el pánico; su problema quedará resuelto, para algo tiene usted un manual en sus manos. Y no le daremos una niña dos propuestas de solución. Pero, ¿dónde y cómo se produce este error?. Quizás no tuvo hasta ahora problema alguno en la utilización de las órdenes MERGE y CHAIN MERGE. Puede ser. De hecho, en rutinas cortas es relativamente improbable que aparezca un error.

Compruébelo mediante nuestro ejemplo:

NEW

```
10 PRINT x;"Ha funcionado."
20 GOTO 20
```

SAVE "dos"

NEW

```
10 PRINT "Lo intentamos..."
20 x=1
30 CHAIN MERGE "dos"
```

RUN

Tras ello puede LISTar el programa. Todo como se quería y esperaba. (Ver también el capítulo correspondiente en el manual de BASIC CPC). Introduzca ahora lo siguiente:

NEW

```
26 PRINT x;"Ha funcionado"
27 END
```

SAVE "dos"

NEW

```
10 PRINT "Lo intentamos..."
20 x=1
25 CHAIN MERGE "dos"
```

RUN

Pero atienda a los números de linea; son importantes en nuestro ejemplo.

Tras poner en funcionamiento el programa, obtendrá un

EOF met in 25

Sabé que EOF significa End of File, y se preguntarà, no sin razon, còmo rayos puede el ordenador tropezar en un End of file. Exactamente éste es el error en el sistema operativo. Hemos incitado la apariciòn de este error al incluir en nuestro ejemplo el nùmero de linea 26. Cuando, al volver a cargar, se halla un &ia (hexadecimal) = 26 (decimal), usted obtiene el mensaje de error EOF met. El còdigo 26 no aparecerà en un programa como caràcter ASCII, pero sì en la codificaciòn de programas.

In el almacenamiento en disco de un programa se escribe, para cada linea del programa, su longitud y direcciòn inicial en memoria. Supone tres caracteres por linea, y un riesgo muy grande de que uno de estos tres bytes resulte ser &ia. Ademàs, el nùmero de linea se almacena como nùmero entero en formato de 16 bits. Tambièn aquí puede aparecer un &ia (como en nuestro ejemplo). La probabilidad es por tanto de $1:(256/5) = 1:51,2$ por linea, lo que significa que, estadisticamente, tras 52 lineas, habrà aparecido un "EOF". Pero en la pràctica suele bastar con menos lineas.

Un modo de solucionar esto es almacenar los programas como ficheros ASCII. Con ello, los programas ya no son codificados sino que se escriben caràcter a caràcter, tal y como ocurre en la pantalla. Se produce un alargamiento no crítico de los programas; un programa de 30 K pasa a ocupar unos 32 K. En nuestro ejemplo deberia haber introducido lo siguiente:

SAVE "dos",a

La ,a significa fichero ASCII. Esta primera soluciòn es tambièn la màs sencilla. Pero si tiene programas protegidos que desea volver a cargar (por ejemplo), esta soluciòn le sirve de bien poco.

Como alternativa podemos ofrecerle el siguiente programa, que ejecuta la orden CHAIN-MERGE tal y como conocemos del BASIC de cassette y está descrito en el manual.

```
1'
2'Firmware-Patch para CHAIN-MERGE
3'Amstrad CPC & DDI-1
4'
5'
6'
10 MEMORY HIMEN-41
20 DEF FNmsb(a)=&FF AND INT(a/256)
30 DEF FN1sb(a)=&FF AND UNT(a)
40 FOR i=HIMEN+1 TO HIMEN+38
50 READ byte
60 POKE i,byte
70 NEXT i
80 POKE HIMEN+3,FN1sb(HIMEN+39)
90 POKE HIMEN+4,FNmsb(HIMEN+39)
100 POKE HIMEN+9,FN1sb(HIMEN+41)
110 POKE HIMEN+10,FNmsb(HIMEN+41)
120 POKE HIMEN+18,FN1sb(HIMEN+1)
130 POKE HIMEN+19,FNmsb(HIMEN+1)
140'CAS IN CHAR
150 POKE HIMEN+39,PEEK(&BCB0+0)
160 POKE HIMEN+40,PEEK(&BCB0+1)
170 POKE HIMEN+41,PEEK(&BCB0+2)
180 &BCB0+0,&C3
190 POKE &BCB0+1,FN1sb(HIMEN+1)
200 POKE &BCB0+2,FNmsb(HIMEN+1)
210 DATA &e5,&2A,&00,&00,&22,&B0,&bc
220 DATA &3a,&00,&00,&32,&B2,&bc
230 DATA &cd,&B0,&bc,&21,&00,&00
240 DATA &22,&B1,&bc,&21,&B0,&bc
250 DATA &36,&c3,&e1,&d8,&c8,&fe,&1a
260 DATA &37,&3f,&c0,&b7,&37,&c9
```

Una vez tecleada esta rutina, deberia almacenarla en disco. Cuando tenga un programa que utilice el comando CHAIN-MERGE, deberia poner esta rutina al principio de dicho programa. Tras la ejecucion de la rutina puede borrar las lineas correspondientes mediante DELETE (tambien posible en el programa). Aqui solo se puede decir de nuevo lo que es necesario en todas las diferencias de los distintos ordenadores Amstrad "compatibles": en caso de que sea poseedor de un CPC 664 ó de un CPC 6128, tiene la suerte de no tener este fallo en su sistema operativo, pero: en caso de que desee escribir software, que pueda luego ser usado por la mayor cantidad de gente, ha de situarse siempre en el caso mas desfavorable, por motivos de compatibilidad, que en este caso es el del CPC 464 con sus debilidades.

5.2 Mensajes de error

El presente capítulo se ha de ocupar de los mensajes de error de disco, que, de tanto en tanto, ha de enviar el ordenador Amstrad. Dado que tanto el CPC 664 como el CPC 6128 ya ofrecen una posibilidad de intercepción de mensajes de error (aunque bien poco cómoda), presentaremos aquí un programa ejemplo para el CPC 464. En cualquier caso, en el capítulo 5.3 hay también una rutina de intercepción de errores para el CPC 664 y el CPC 6128. El esquema de esta rutina de intercepción de errores es la misma para los tres ordenadores; a pesar de ello se recomienda el estudio del presente capítulo.

Prácticamente todos los usuarios reaccionan ante la aparición mensajes de error frunciendo el ceño, ya que les recuerdan los desesperados intentos llevados a cabo para proteger su programa de tales mensajes de error de la unidad de discos.

Los mensajes de error son algo inherente al trabajo con ordenadores. Todos cometemos errores, y el ordenador los muestra en pantalla. De igual modo, el floppy tampoco acepta todo lo que le es transferido. Si se intenta, por ejemplo, cargar un fichero inexistente, su lapidaria observación será:

Filename.Ext not found

Sabrá entonces que, o bien introdujo incorrectamente el nombre de fichero, o bien insertó un disco equivocado. La unidad de disco se tomó el mayor empeño en servirle antes de enviar el mensaje de error. Si no se da extensión (tipo de fichero) alguna, el floppy DDI-1 realizará tres intentos para hallar el fichero deseado.

- 1) Como Filename.
- 2) Como Filename.BAS
- 3) Como Filename.BIN

Sólo tras verificar error en los tres intentos le dará el floppy el citado mensaje de error. En caso de que indique la extensión, tan sólo se realiza un intento. Ejemplos al respecto serían LOAD "archivo.bas" u OPENIN "datos.dat", etc.

En el capítulo 1.5 se describió el modo de evitar algunos errores. Pero no siempre pueden aplicarse fructíferamente estas ideas de programación.

Existen errores como Disc Missing (Falta disco), entre otros, imposibles de evitar por programación.

Para que ello ocurra, no es indispensable que la unidad esté vacía. Con que el disco no esté totalmente centrado, ya obtendrá el mensaje de error siguiente:

Drive A: Disc Missing

Retry, Ignore or Cancel?

Otro ejemplo; cuando su disco ya no esté en buenas condiciones, y aparezca un Read Fail (Fallo de lectura) como mensaje de error. La lista de posibilidades podría alargarse a voluntad. Ello no sería tan grave, y llegaría incluso a constituir una nueva ventaja del floppy si no existiera una traba decisiva. Si durante el transcurso de un programa el floppy detecta un error, sea cual fuere, el texto de diagnóstico de error aparece en pantalla, y el programa es inmediatamente interrumpido. No es tan grave -dirá usted-, a mí también me pasa cuando tengo un error en el programa. Y es cierto, pero:

Los programas pueden depurarse casi totalmente. Pueden, por ejemplo, descartarse totalmente los errores de sintaxis y comprobar e interceptar entradas erróneas por parte del usuario. De este modo pueden interceptarse prácticamente todos los errores imaginables durante el transcurso de un programa.

Además resulta posible utilizar el comando BASIC ON ERROR GOTO para que, ante un mensaje de error durante el transcurso del programa, éste no se destruya sino que simplemente se interrumpa.

Así, ante un error se interrumpe el normal transcurso del programa, la posición se memoriza temporalmente y el programa salta a una subrutina que eventualmente puede reaccionar ante dicho error. Una programación de este tipo puede interceptar una interrupción del transcurso del programa al 100%.

Pero ¿qué haremos para averiguar, en el programa, si hay o no un disco en la unidad? Puede informarse al usuario del programa que inserte el disco, pero ésto no representa sino que una de las posibles causas de error, y, por tanto, de mensajes. Bueno, dirá usted, usemos el comando ON ERROR GOTO que es realmente práctico.

Pero lo malo es que ON ERROR GOTO no tiene efecto alguno ante mensajes de error provenientes del floppy. A pesar de ello, los mensajes de error siguen siendo enviados y el programa interrumpido.

(Aquellos de ustedes que todavía no conozcan suficientemente el comando deberían estudiarlo en el manual.)

Llegados a este punto hay que observar lo siguiente: el Amstrad CPC 664 y el CPC 6128 ofrecen la posibilidad de interceptar mensajes de error mediante el comando ON ERROR GOTO. La variable de error ERR contiene entonces el valor 32. La variable de error DERR contiene además el número de error suministrado por el DOS. Así que el programa no se

interrumpe, pero el texto de error aparece en pantalla. Otra desventaja adicional es que no se puede preguntar directamente de qué error se trata, dado que ERR contiene siempre el valor 32 y que la variable DERR tampoco es demasiado explícita al respecto.

Aquí que un buen consejo en este tema sería de agradecer. Hasta el más curtido programador convendrá en ello. Puede imaginarse lo desagradable que resulta haber introducido 200 juegos de datos, disponerse a almacenarlos en disco, y que el floppy dé el siguiente mensaje:

Disc Full

El programa se interrumpe, y usted puede volver a empezar a introducir sus 200 juegos de datos. Un programa tal no ofrece precisamente un aspecto profesional. La primera obligación de la programación profesional es salvaguardar los programas de la aparición de errores. Y el floppy es muy dado a los errores.

En este capítulo hallará una rutina en lenguaje máquina que relegará este problema a una cuestión del pasado. Los errores ya no son llevados a la pantalla y los programas no se interrumpen. Se hubiera podido hacer de modo que el programador no hubiese notado en absoluto el envío de un mensaje por parte del floppy. Pero ello no es deseable, ya que el conocimiento del error no es problemático, lo único que interesa es que en ello no se interrumpa el programa. En esta rutina el problema se resolvió de modo que, ante un mensaje de error, la unidad de disco envíe el mensaje

Unknown user function

Este mensaje aparece de hecho en muy raras ocasiones, por lo que fue elegido para distinguir en el programa los errores de floppy de otros errores. Todos los mensajes de error tienen sus códigos correspondientes, que pueden ser llamados

en las variables estàndar ERR. Tambièn podemos preguntar por la linea en la que apareciò el ùltimo error. Para ello se utiliza la variable estàndar ERL.

Asì que cuando aparece un mensaje de error del floppy, es interceptado y temporalmente almacenado en una memoria intermedia especial. El programa no se interrumpe y se genera un error UNKNOWN USER FUNCTION, de còdigo de error 18. Este puede ser perfectamente interceptado mediante la orden ON ERROR GOTO. Cabe ahora buscar en la rutina de error correspondiente què error fue modificado. Si la variable de còdigo de error ERR vale 18, el error proviene del floppy. Puede transferirse el texto a una variable de cadena mediante la rutina para determinar exactamente el error. En el programa puede actuarse consecuentemente segùn el error aparecido. Resulta posible activar o desactivar la rutina en cualquier momento.

Un programa a titulo de ejemplo

```
10 ON ERROR GOTO 1000
20 CALL ein: 'rutina de error activa
30 OPENIN "cifras"
40 WHILE NOT EOF
50 INPUT #9,a
60 PRINT a,
70 WEND
80 CLOSEIN
90 CALL aus : END
1000 IF ERR<>18 THEN RESUME NEXT
1010 ds$="+":CALL msg,@ds$
1020 PRINT "Disk: ";ds$
1030 RESUME 90
```

Reconocerà que las rutinas de error son llamadas mediante la orden CALL. Existen tres direcciones distintas de llamada:

- Call ein Mediante Call ein activa usted la rutina de intercepción de errores. Se desvía una gran cantidad de vectores del sistema operativo. De otro modo no puede resolverse el problema. A partir de este momento ya no aparecen en pantalla los mensajes de error, sino que se interceptan y almacenan en una memoria intermedia. Tras un mensaje enviado por el floppy se genera un Unknown User Function. Tal error puede ser interceptado mediante ON ERROR GOTO
- Call msg,a\$ Se transfiere el último mensaje de error notificado a la variable de cadena a\$ (cualquier otra variable de cadena resulta igualmente válida). Preste atención a que la variable a\$ haya sido previamente declarada, basta con un a\$=""; en caso contrario obtendría un IMPROPER ARGUMENT(argumento. impropio).
- Call aus Se desconecta de nuevo la rutina de intercepción de errores. A partir de ahora vuelven a aparecer en pantalla los textos de error. Se recomienda volver a desconectar las rutinas al fin de cada programa, ya que en caso contrario se asombrará cuando un FILE NOT FOUND no fuera notificado como de costumbre.

!En la utilización de estos comandos CALL, se da por supuesto que las rutinas se hallan en memoria!

Explicación del programa de ejemplo:

Línea	Explicación
10	Se define que tras un mensaje de error el programa prosigue en la linea 1000.

- 20 Se activa la rutina de error. A partir de este momento, los mensajes de error ya no son mostrados en pantalla por el floppy sino que serán interceptados y temporalmente almacenados.
- 30 El fichero secuencial "cifras" es abierto. En caso de que no exista, se genera un "File not found".
- 40-80 Si el fichero pudo ser hallado, se leen las cifras y se les da salida.
- 90 La rutina de intercepción de errores vuelve a ser desconectada y finaliza el programa.
- 1000 Aquí se comprueba si se trata de un error debido al floppy. En caso negativo, el programa prosigue inmediatamente tras el error.
- 1010 Se declara la variable ds\$. Ello es necesario para no obtener "Improper argument" en @ds\$. Tras ello se transfiere el texto de error proporcionado a la variable ds\$.
- 1020 El texto transferido es editado a pantalla. Esto es únicamente una de las muchas aplicaciones posibles.
- 1030 El programa prosigue en la línea 90.

Esto como ejemplo de utilización. Otro ejemplo de cómo trabajar muy provechosamente con la rutina de errores lo constituye el programa de gestión de ficheros del presente capítulo.

Se ha mostrado como muy práctico y útil el crear y comprobar en el programa los llamados flags -banderas o semáforos-. Se puede reconocer en el transcurso del programa si ha aparecido o no un error y reaccionar en consecuencia.

ello podria tener en el ejemplo el siguiente aspecto:

```
10 ON ERROR GOTO 1000
20 CALL ein : 'rutina de error activa
30 errflg=0 : OPENIN "cifras" : IF errflg THEN 100
40 WHILE NOT EOF
50   INPUT #9,a
60   PRINT a,
70 WEND
80 CLOSEIN
90 CALL aus : END
100 IF RIGHT$(ds$,9)="not found" THEN REM reaccion
110 PRINT "Disk -- ";ds$
120 END
1000 IF ERR<>18 THEN RESUME NEXT
1010 ds$="+": CALL msg,@ds$
1020 errflg=1
1030 RESUME NEXT
```

En este ejemplo puede ver que tras la instrucción OPENIN se comprueba si existe un error. Si no existe error alguno, prosigue el transcurso normal del programa.

Pero si existe algún error, se comprueba si se trata de un "File not found". Si efectivamente lo es, se puede reaccionar en consecuencia (por ejemplo modificando el nombre de fichero). En caso contrario se da salida al mensaje de error. Naturalmente pueden hacerse aquí todavía otras consideraciones. Ante un Drive A: Disc Missing se le podria pedir al usuario, en lenguaje comprensible, que inserte un disco.

Con este modo de tratamiento de errores se puede determinar mediante una apertura de prueba de un fichero si este fichero ya existe, y abrirlo en caso de que sea necesario hacerlo.

Como ve, las posibilidades son múltiples y la utilización sencillísima.

Puede ubicar la rutina de intercepción de errores en cualquier zona de memoria. Està impresa en un listado assembler para aquellos de entre ustedes que programen en lenguaje máquina. Para los demás, hay un cargador BASIC para situar la rutina en lenguaje máquina en cualquier zona. La rutina ocupa 229 bytes (únicamente!). Realmente un gasto mínimo para la utilidad que tiene. En la linea 70 del cargador BASIC puede poner la dirección de inicio que ha de tener el programa. Si tiene un programa en el que no aparezca ningún comando SYMBOL AFTER, puede ubicar su rutina en la zona superior de memoria para restar la mínima memoria al BASIC.

En el BASIC de cassette dispone usted de 43533 bytes para sus programas y datos. El límite superior de la memoria se halla en &AB7F. Cuando conecta la unidad de discos se ocupa algo de memoria para el DOS y los buffers de entrada y salida. Tras la conexión HIMEM està en &A67B y usted dispone todavía de 42249 bytes. Pero el DOS precisa aún 4096 bytes para los ya mencionados buffers de entrada y salida. Estos son, en cualquier caso, reubicables (al contrario que la memoria de sistema). Situamos esta memoria mediante:

```
OPENOUT "dummy"  
MEMORY HIMEM-1  
CLOSEOUT
```

Ahora obtenemos para HIMEM el valor &967A. Quedan aún 38152 bytes libres. Así que en el caso más favorable el límite superior para BASIC està en &967A. Nuestra rutina tiene una longitud de 229 bytes y necesita, además, 40 bytes como buffer para el texto del mensaje de error. Así que calculamos &967A - 229 - 40 = &956D y hacemos que nuestra rutina comience en &956D. Ponga este valor en la línea 70 del cargador.

Si utiliza el comando Symbol After, ha de ubicar la rutina en la memoria a mayor profundidad. Para ello desconecte y vuelva a conectar su ordenador. Luego, introduzca

Symbol After n

Introduzca, en lugar de n, el mayor valor que aparezca en el programa. Despu s, introduzca

```
OPENOUT "dummy"  
MEMORY HIMEM-1  
CLOSEOUT
```

```
PRINT HEX$(HIMEM-229-40)
```

Obtendr , con Symbol After 190 por ejemplo, el resultado de &93DD. Defina &93D0 como direcci n inicial. Eso es todo.

Cree de este modo, y de momento para practicar, una rutina ubicada en &9650. Podr  utilizarla en la mayoria de programas, mientras no utilicen comando symbol after alguno y no incluyan otras rutinas en c digo m quina. Luego, almacene en disco este fichero, dado que es m s sencillo leer la rutina de intercepc n de errores del disco en programas, que encadenar cada vez todas las l neas DATA al programa. Si desea utilizar por fin la rutina de intercepc n de errores en un programa, observe el siguiente orden:

- 1) Creaci n del fichero binario de programa mediante el programa cargador BASIC y almacenamiento del programa en disco (por supuesto en el disco en el que est  almacenado el programa que precisa y carga esta rutina).

- 2) Intercalar la rutina en el programa. También aquí es importante el orden en que ello sucede ya que en caso contrario pueden perderse innecesariamente 4096 bytes. Ello ocurre cuando en primer lugar se carga la rutina, se rebaja luego el límite superior de memoria y sólo entonces se define el buffer.

Un programa en el que se leyera la rutina de intercepción de errores debería tener un aspecto similar a este:

```
10 ON ERROR GOTO 20000 'rut. intercepción
20 LOAD "ERROR.BIN"
30 MEMORY &9650-1
40 OPENOUT "dummy"
50 MEMORY HIMEM-1
60 CLOSEOUT
70 .
80 .
90 .
```

Ya reconoce el orden. En primer lugar ha de generarse la rutina y almacenarla en disco bajo el nombre "error.bin"; por descontado que resulta posible cualquier otro nombre, que puede elegirlo en el cargador.

El programa cargarà en primer lugar este fichero, el límite de memoria será después fijado justo delante de nuestra rutina, para evitar que dicha rutina sea sobreescrita por variables o por los buffers del DOS. Sólo entonces se ubican los buffers del DOS y se rebaja de nuevo el límite. Ahora ya no puede ser sobreescrita. Lo más práctico antes de utilizar programas de este tipo es que desconecte y vuelva a conectar el ordenador, ya que el haber tenido previamente dispuesto el buffer del DOS o el límite rebajado de algún modo puede producir complicaciones.

Tras todo ello lo mejor es definir cuatro variables más que puede precisar en el programa en cualquier momento.

MSG=▲+3
AUS=╝+6
DS\$=""

EIN = BASE + 0

MSG = BASE + 9

AUS = BASE + 6

Los valores, evidentemente, han de ser consecuentemente cambiados para otra dirección inicial. Se trata de las direcciones de llegada para las distintas rutinas. Le deseamos mucho éxito en la utilización de estas rutinas. Pero antes de que se lance a teclear el cargador BASIC: en el presente capítulo se halla otro paquete de rutinas que posibilitan el almacenamiento relativo de datos. Si desea usar almacenamiento relativo de datos, mejor teclee el cargador BASIC del almacenamiento relativo de datos; lo contrario sería perder demasiado tiempo.

Los usuarios del CPC 664 y del CPC 6128 que encuentren innecesaria la rutina de errores porque su BASIC ya la incorpora, deben tener en cuenta lo siguiente:

- 1) Los textos de error aparecen en pantalla; ello no es muy cómodo para programas profesionales.
- 2) No tiene compatibilidad con los CPC 464.

- 3) Consultando la variable ERR puede determinarse la existencia de un error de floppy, pero no se obtiene el mensaje de error concreto, ni siquiera utilizando la variable DERR. No se puede pues diferenciar un DISC-FULL del mensaje de error DISC IS MISSING.

5.2.1 Comentarios a la rutina de intercepción. ¿Cómo se usa?

Para interceptar mensajes de error se precisan algunos conocimientos del interior del CPC, ya que ha de llevarse a cabo una incursión nada despreciable a la vida íntima del sistema operativo. Retomaremos la posibilidad del CPC de desviar ciertos vectores. Haremos frecuente uso de ello.

Cuando el floppy envía a pantalla un mensaje de error, se efectúa una llamada a la rutina del sistema operativo TXT OUTPUT en &BBA5. Esta rutina sirve para dar salida a pantalla a un carácter en el acumulador, a la ventana actual. Esta rutina es desviada, parcheada en el argot.

Así que la rutina de intercepción se activa y ejecuta antes que la rutina TXT OUTPUT normal, y ello con cada carácter que ha de aparecer en pantalla. En esta rutina se comprueba si el carácter a imprimir proviene del floppy. Ello se realiza comprobando en el stack (la pila) la dirección de retorno de salto. La ROM de floppy se halla en la zona superior a &C000. También se halla aquí el sistema operativo del BASIC. La rutina de transmisión de mensajes de error se halla en la zona &CB00-&CBFF de modo que simplemente hemos de consultar el byte alto de la dirección de retorno del salto. Así que si se llama desde esta zona a TXT OUTPUT, se trata de un carácter del floppy, que será interceptado por la rutina y temporalmente almacenado para su posterior uso.

Al mismo tiempo se activa un flag para poder comprobar posteriormente si apareció un error. El flag sirve además para interceptar el mensaje BREAK. Y es que éste no proviene

de la ROM de floppy, sino de la ROM de BASIC "normal". Todas estas cosas se llevan a cabo en la rutina OUTCHK (ver listado Assembler). En esta rutina se parchean (desvian) los vectores, en la rutina RESET se les devuelven sus valores normales.

Cuando el floppy ha dado salida al mensaje de error, salta a la ROM de BASIC y precisamente a la posición en la que se interrumpen los programas. Se da salida al mensaje BREAK y se interrumpe el programa. Aquí hemos de desviar el vector Ready Modus para evitar que puedan interrumpirse los programas. En esta rutina se comprueba si está activado el flag de error, ya que se puede llegar con ello al Ready Modus sin aparición de mensaje de error. Si el flag de error está activado, se genera el mensaje de error #18 (Unknown User Function), que puede ser posteriormente interceptado.

Tenemos finalmente la rutina GETTXT que sirve para transmitirnos al BASIC el mensaje de error, de modo que obtenemos el mensaje en texto comprensible, contenido en una variable de cadena cualquiera.

La rutina KWC sirve para enviar a la pregunta

Retry, Ignore or Cancel

una C por Cancel. De este modo se reconoce inmediatamente el error, y se interrumpe el desarrollo del programa.

L I A T E N C I O N - I M P O R T A N T E !!

Si da salida al directorio de un disco, estos caracteres se interpretan también como un error, dado que provienen exactamente de la misma zona de memoria que los mensajes de error. Ello es válido tanto para la orden IDIR como para la CAT. ¡Cuando la rutina de errores está conectada, la orden

IDIR no funciona! Si desea utilizar este comando use:

```
CALL aus : IDIR : CALL ein
```

En la rutina se trató la orden CAT de un modo especial. También sus vectores fueron desviados, de modo que el comando CAT ha de usarse como siempre.

Cargador BASIC de rutina intercepción errores para CPC 464:

```
10 *****
20 **** Rutina de intercepcion de errores *****
30 *** (C) 1985 by DATA BECKER GmbH JS 22/3/85 *
40 *****
50 '
60 DEFINT a-z
70 adresse = &A000 : 'Direccion inicial para rutina
80 :
90 DATA &C3,&09,&51,&C3,&B3,&51,&C3,&AC,&51,&3E,&C3,&32,&01,
&AC,&32,&5A
100 DATA &BB,&32,&06,&BB,&32,&9B,&BC,&21,&60,&51,&22,&02,&AC
,&21,&30,&51
110 DATA &22,&5B,&BB,&21,&74,&51,&22,&07,&BB,&21,&D1,&51,&22
,&9C,&BC,&C9
120 DATA &E3,&F5,&7C,&FE,&CB,&20,&1E,&3E,&01,&32,&E3,&51,&F1
,&E5,&2A,&E4
130 DATA &51,&FE,&0A,&28,&0B,&77,&23,&7D,&FE,&28,&20,&01,&2B
,&22,&E4,&51
140 DATA &E1,&F5,&F1,&E3,&C9,&3A,&E3,&51,&B7,&20,&F7,&F1,&E3
,&CF,&00,&94
150 DATA &3A,&E3,&51,&B7,&CB,&AF,&32,&E3,&51,&21,&E6,&51,&22
,&E4,&51,&1E
```

```
160 DATA &12,&C3,&94,&CA,&F5,&3A,&E3,&51,&B7,&20,&04,&F1,&CF
, &3C,&9A,&F1
170 DATA &3E,&43,&C9,&1E,&02,&FE,&01,&C0,&DD,&5E,&00,&DD,&56
, &01,&OE,&FF
180 DATA &21,&E6,&51,&7E,&FE,&0D,&23,&28,&FA,&2B,&E5,&2B,&0C
, &23,&7E,&FE
190 DATA &0D,&20,&F9,&79,&12,&E1,&EB,&23,&73,&23,&72,&C9,&3E
, &MC9,&32,&01
200 DATA &AC,&3E,&CF,&32,&5A,&BB,&32,&06,&BB,&3E,&DF,&32,&9B
, &MC9,&21,&00
210 DATA &94,&22,&5B,&BB,&21,&3C,&9A,&22,&07,&BB,&21,&8B,&A8
, &72,&9C,&BC
220 DATA &C9,&F5,&E5,&CD,&AC,&51,&E1,&F1,&CD,&9B,&BC,&E5,&F5
, &CD,&09,&51
230 DATA &F1,&E1,&C9,&00,&E6,&51
240 :
250 FOR i=adresse TO adresse+&E5
260 READ a
270 POKE i,a
280 s=s+a
290 NEXT
300 IF s<>28065 THEN PRINT CHR$(7) "*** Error en DATAS ***" :
310 IND
320 :
330 DATA &01,&09,&04,&B3,&07,&ac,&18,&60,&1e,&30,&24,&74,&2a
, &d1,&3a,&e3
340 DATA &3f,&e4,&4e,&e4,&56,&e3,&61,&e3,&67,&e3,&6a,&e6,&6d
, &e4,&76,&e3
350 DATA &91,&e6,&d4,&ac,&de,&09,&e4,&e6
360 :
370 FOR i=1 TO 20
380 READ of1,of2
390 ad1=adresse+of2
400 POKE adresse+of1,ad1 AND 255 : 'Byte bajo
410 POKE adresse+of1+1,INT(ad1/256) AND &FF
420 NEXT
430 :
440 INPUT "Quiere almacenar el fichero ? (S/N) ",a$
450 IF UPPER$(a$)="S" THEN INPUT "Nombre de fichero : ",b$ :
460 GIVE b$,b,adresse,&E8
```

5.3 Gestión de ficheros relativos

En el capítulo 1.5 aprendió la programación de ficheros secuenciales, y estará ya interesado en la gestión de ficheros relativos que brinda realmente algunas ventajas dignas de ser consideradas. Este modo de programación no es, de todos modos, el óptimo en cualquier caso: especialmente en utilidades menores resultaría frecuentemente sin sentido su utilización, ya que ocuparía más tiempo y espacio que la programación secuencial. La gestión de ficheros relativos es interesante, cuando hay que realizar frecuentes correcciones y accesos al fichero. Aparte de las distintas exigencias que ofrecen al usuario los almacenamientos relativo y secuencial, existen otras diferencias agravantes.

En primer lugar, ya no es necesario leer completamente un fichero, pasándolo a la memoria, para poder elaborarlo. También puede olvidarse de los complicados bucles para la búsqueda o lectura en el fichero de un juego de datos en concreto. Existen nuevas reglas para ello, por las que nos hemos de regir: hay que reflexionar detenidamente cuál será la longitud máxima de un juego de datos (¿cuántos caracteres tiene nuestro juego de datos?), y qué número de juegos de datos contendrá nuestro fichero (¿cuántos juegos de datos queremos elaborar?). Hasta ahora, en la gestión secuencial de fichero, podía prescindir de esta reflexión previa - si bien cualquier programador serio valorará siempre la magnitud de sus ficheros desde el principio, aunque sea por el simple motivo de tener una idea de la cantidad de juegos de datos que caben en un disco.

Efectuemos un cálculo de este tipo de la mano de nuestro ejemplo del capítulo 1.5 - recordará nuestra agenda telefónica, con los tres campos de datos:

Campo 1) Apellido
Campo 2) Nombre
Campo 3) Número de teléfono

Supongamos que deseamos almacenar 50 números de teléfono tales; tendremos, por tanto, 50 juegos de datos.

Para poder admitir en su caso más números de teléfono, establecemos 100 juegos de datos. Hemos de determinar aún la longitud de los juegos de datos. Ello se lleva a cabo asignando a cada campo de datos una longitud máxima, que no puede ser sobrepasada en el programa. Así que debe vigilar que este límite sea respetado ya que lo contrario surgen complicaciones. Esto no es una chapuza, ni resulta incómodo; hallará esta limitación en toda gestión de ficheros relativos, incluso en los mayores ordenadores. La longitud de juego de datos es un elemento importante para el desarrollo sin conflictos de los cálculos en los que hay que hallar un determinado juego de datos.

En "nuestra" gestión de ficheros relativos dispone usted incluso de un cierto margen: puede sobrepasar los límites de campos de datos individuales, mientras mantenga correspondientemente recortado otro campo de datos. Ello se hizo para estructurar con una cierta flexibilidad la gestión de ficheros, pero no en todos los sistemas es así. Sólo ha de vigilar que no se sobrepase la longitud conjunta. Pero volvamos a nuestro ejemplo:

Ahora hemos de asignar las longitudes máximas, es decir, el número máximo de caracteres que puede tener cada campo de datos. Tomemos, por ejemplo, la siguiente asignación:

	Apellido: 25 caract.
+	Nombre: 15 caract.
+	Teléfono: 15 caract.
+	Separadores: 3 caract.
<hr/>	<hr/>
=	Todal: 58 caract.

Así que el apellido puede tener un máximo de 25 caracteres, lo que debería bastar incluso para apellidos largos o compuestos, el nombre de pila está limitado a 15 caracteres. Por último tenemos hasta 15 caracteres previstos para el tercer campo, el número de teléfono. En total suman 55 caracteres por juego de datos. A estos 55 hemos de sumar otros tres caracteres, ya que tenemos tres campos de datos que han de ser separados entre si. El carácter de separación no resultaría necesario si los campos de datos tuvieran longitud fija, pero dado que no es éste el caso, también en "nuestra" gestión de ficheros relativos precisamos un carácter de separación. De este modo llegamos a una longitud de

58 caracteres.

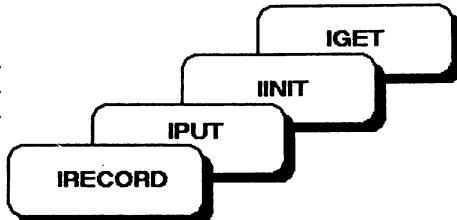
Para no alargar demasiado el listado de la rutina para la gestión de ficheros relativos - de hecho, ha de teclearla - se estableció la siguiente condición: la longitud del juego de datos ha de ser redondeada a la siguiente potencia de 2. Ello significaría en nuestro ejemplo que dariamos una longitud de juego de datos de 64.

La longitud máxima de juego de datos es de 512 bytes. Así que puede elegir entre las siguientes longitudes:

2, 4, 8, 16, 32, 64, 128, 256 y 512.

Dispone con ello de nueve longitudes de juego de datos distintas, entre las que elegir. Si su longitud calculada resulta ser de 32, está de suerte, ya que puede dar estos 32 como longitud. En caso de que fuera incluso un único carácter más, debería decidir si elegir una longitud de juego de datos de 64, o replantearse si hay algún campo que pueda ser recortado en un carácter. En nuestro ejemplo calculamos una longitud de 58. La diferencia entre 64 y 58 es 6; usted tendrá 6 bytes de más - decir que los desperdicia sería exagerado - por juego de datos. Puede ajustarse a esta longitud de juego de datos, y alargar éste o aquel campo en algún que otro carácter. Esta limitación no es de las que le quitan a uno el sueño. En la gestión de ficheros relativos puede olvidarse tranquilamente de todos los comandos utilizados hasta ahora para la gestión de ficheros secuenciales, a excepción de los comandos OPENIN y CLOSEIN. Tenemos en compensación cuatro nuevos comandos, que son:

IINIT,<longitud>
IRECORD,<número de record>
IPUT,<lista de parámetros>
IGET,<lista de parámetros>



El comando GET es comparable al comando LINE INPUT, es decir, sirve para la lectura de datos; PUT sirve para la escritura de datos, siendo por tanto comparable a PRINT.

Si un campo de datos ha de contener números, éstos han de ser convertidos en cadena mediante el comando STR\$. En números reales la longitud del campo es 12, para enteros sólo 9.

Dado que en el floppy DDI-1 no disponemos en principio de ficheros relativos, éstos son simulados mediante ficheros secuenciales. Antes de poder trabajar con un fichero relativo, éste ha de ser "creado". Esto se realiza creando un fichero secuencial prácticamente vacío. Recordemos que nuestro fichero ejemplo ha de contener 100 juegos de datos de una longitud de 64 caracteres. Precisamos por tanto de un espacio efectivo de memoria de $100 \times 64 = 6400$ bytes. La creación de un fichero de este tipo es muy sencilla:

```
10 REM =====
20 REM      Crear fichero REL
30 REM =====
40 :
50 OPENOUT "fichero.rel"
60 FOR i=0 TO 100
70 PRINT #9, STRING$(64,13);
80 NEXT
90 CLOSEOUT
```

El número previsto de juegos de datos ha de ser incorporado en la línea 60; tal número era 100 en nuestro ejemplo. En la línea 70 se crea el espacio para cada juego de datos; lo que hacemos es llenar nuestro fichero (aún) secuencial sólo con el carácter de <Carriage Return>. (Puede elegir también cualquier otro carácter, pero resulta aconsejable el carácter separador <Carriage Return>, ya que en caso contrario, y bajo ciertas circunstancias, podrían surgir problemas). No resulta imprescindible cerrar la linea 70 con un punto y coma, pero de este modo el cálculo resulta exacto. Con estas pocas líneas tenemos suficiente, hemos creado nuestro fichero REL, que ya puede ser elaborado.

Por cierto: En caso de que su juego de datos sea mayor que 255 bytes, no resulta posible expresar esa longitud de juego de datos en un comando STRING\$. Divida entonces su longitud de juego de datos en dos comandos STRING\$, como, por ejemplo, para la longitud de datos:

```
256 STRING$(128,13);STRING$(128,13);
512 STRING$(200,13);STRING$(200,13);STRING$(112,13);
```

Es una pena, pero no funciona de otro modo, ya que el sistema operativo no lo permite para el comando STRING\$.

Por otra parte, ha de vigilar que la longitud del fichero no excede 131072 caracteres (es decir 2^{17}), ya que fallan los cálculos para ficheros mayores.

Ahora podemos escribir en nuestro fichero 100 juegos de datos y cada juego tendrá en adelante un número que lo determinará únicamente. En la jerga normalmente empleada, también se llama RECORD o registro al juego de datos, de modo que cada juego de datos puede determinarse mediante su número de Record. Al primer Record le corresponde el número de Record 0.

Al abrir un fichero REL puede utilizar, antes o después, el comando OPENIN. Un fichero REL abierto puede ser leído y escrito, pero para la apertura siempre será OPENIN.

Mediante el comando IINIT se determina la longitud de record, es decir, la longitud de un juego de datos cualquiera. Esto debería ocurrir inmediatamente tras la apertura del fichero, OPENIN e IINIT van siempre juntos, como si fuesen siameses. El comando IINIT tiene un parámetro; puede usted utilizar tanto constantes (que será lo más normal) como variables. En nuestro ejemplo tendría usted que dar entrada a lo siguiente:

```
IINIT,64
```

Pero tambièn cabria proceder asi:

```
longitudrec=64 : IINIT,longitudec
```

Mediante el comando IRECORD determina usted el proximo record que ha de ser elaborado; es indiferente si ello va a consistir en una lectura o en una escritura. El primer record, ésto es, el primer juego de datos tiene el nùmero de Record 0 - si no quiere acostumbrarse a esta secuencia, puede dejar este primer Record "abandonado" o utilizarlo como memoria especial. El comando IRECORD tiene tambièn un paràmetro que puede ser una constante o una variable. En el caso del comando IRECORD lo más frecuente es una variable. Para posicionar, por ejemplo, el juego de datos 35 teclee:

```
IRECORD,35
```

Los comandos IGET e IPUT son, en sintaxis y utilizaciòn, idénticos. El comando IGET recoge datos del sector elegido y el comando IPUT escribe datos en el sector elegido. Los comandos IGET e IPUT pueden tener cualquier nùmero de paràmetro comprendido entre 1 y 32. La ausencia de paràmetros no provoca mensaje de error alguno, pero el comando resulta inoperante. Pueden transferirse exclusivamente variables de cadena. Las constantes que se deseé escribir han de ser previamente transformadas en variables de cadena. Supongamos que deseamos escribir la palabra "caballo" en el Record 23; la secuencia de órdenes necesaria seria:

```
10 OPENIN "fichero.REL"
20 IINIT,64
30IRECORD,23
40 a$="caballo"
50 INPUT,0a$
60 IINIT,0
70 CLOSEIN
```

En este ejemplo podemos reconocer dos particularidades. Antes de cerrar un fichero relativo, ha de dar obligatoriamente el comando IINIT,0 - después puede cerrar el fichero. Ello tiene por finalidad que los datos eventualmente aún existentes en la memoria sean escritos en el disco antes de cerrar el fichero. Observará otra particularidad en la orden INPUT: la cadena no puede ser simplemente puesta tras la coma, ha de ir precedida de 0. Esto no es achacable a los autores del presente libro, sino que se trata de una exigencia del sistema operativo (ver también capítulo 1), que sólo ofrece esta vía de enlace entre el BASIC y el lenguaje máquina. Pero esta "desventaja" no debería molestarle demasiado, uno se acostumbra en seguida a ello. Si desea transferir más de un parámetro en la instrucción INPUT, puede encadenarlos separándolos mediante comas. Si, por ejemplo deseamos escribir además la palabra "gato" en nuestro record:

```
10 OPENIN "fichero.REL"
20 IINIT,64
30IRECORD,23
40 a$="caballo"
45 b$="gato"
50 INPUT,0a$,0b$
60 IINIT,0
70 CLOSEIN
```

Resulta intrascendente el hecho de escribir todos los campos de datos de un Record en una única instrucción INPUT, o mediante varias instrucciones, ya que los punteros internos se encargan de un desarrollo libre de problemas. Con la

misma facilidad con la que escribimos datos, podemos leerlos, y ello mediante el comando IGET.

Aquí puede transferir cualquier número de variables de cadena como paràmetros. Da también igual el leer todos los campos de datos en una linea o en varias. Leamos de nuevo "nuestro" record:

```
10 OPENIN "fichero.REL"
20 IINIT,64
30 a$="" : b$=""
40IRECORD,23
50IGET,0a$,0b$
60PRINT a$,b$
70IINIT,0
80CLOSEIN
```

Si transfiere a la rutina IGET variables de cadena como paràmetros, estas variables han de haber aparecido al menos una vez en el programa; en caso contrario, el intèrprete BASIC se manifiesta con el mensaje de error:

Improper argument

De hecho, esto es todo lo que ha de tener en cuenta. Para no alargar innecesariamente la rutina se renunciò a la mayoria de comprobaciones de error. Así que ha de tener cuidado en no dar números de Record demasiado elevados (aquí pueden suceder cosas imprevisibles), también ha de tener cuidado en no sobrepasar la longitud màxima de juego de datos, ya que en caso contrario se escribiría en el juego de datos consecutivo. Si abriò un fichero REL, no puede abrir ningún fichero de OUTPUT mediante OPENOUT; así que ''sólo'' puede tener abierto el fichero relativo. Pero ello es comprensible ya que en el fichero relativo puede escribir y leer a la vez.

Vigile que no se le olvide disponer y preservar el buffer de entrada/salida al principio del programa mediante el comando

```
OPENOUT "Dummy" : MEMORY HIMEN-1 :  
CLOSEOUT
```

ya que en caso contrario podrían presentarse complicaciones durante el funcionamiento con las órdenes de ampliación de floppy. Si desea trabajar por sí mismo con la rutina tome el programa de ejemplo de la gestión de ficheros relativos. Puede deducir del mismo cómo y en qué orden debería utilizar las distintas órdenes.

No utilice el comando CAT - cierra el fichero sin almacenar en disco datos que podrían resultar necesarios y sobreescribe el buffer.

Si observa estas reglas le sacará un gran redimiento a la rutina. El fichero REL puede extenderse a cualquier medida, a todo el disco en teoría. Los tiempos de acceso son muy cortos, como demuestra el programa ejemplo. Junto a las órdenes para la gestión relativa de ficheros se integraron también las órdenes para la rutina de intercepción de errores (ver capítulo 5.2). El manejo de los comandos es el descrito en el capítulo 5.2. A partir de CALL ein tiene IAN, de CALL aus surge IAUS y de CALL msg sale IERR. Así que tiene tres nuevos comandos:

```
IAN  
IAUS  
IERR,<variable de cadena>
```

Con estas siete nuevas órdenes se cubren todas las insuficiencias del floppy; ya puede crear programas cómodos y libres de errores sin tener que recurrir a complicados recursos. Hallará impresos tanto un listado Assembler como un Cargador BASIC. Puede deducir el cálculo de la dirección en la que ha de hallarse la ampliación de las descripciones

al respecto en el capítulo 5.2. Esta rutina ocupa &34a = 842 bytes. Normalmente (es decir, en programas sin el comando SYMBOL AFTER) puede utilizar &9300 como dirección básica.

Ya que la solución exacta dada al problema puede deducirse del programa en lenguaje máquina, se explicará aquí simplemente cómo se realizó la gestión de ficheros relativos.

5.3.1 Comentarios al programa en lenguaje máquina

Si deseamos tener un fichero relativo, previamente hemos de "disponerlo". Creamos para ello en primer lugar un fichero secuencial con la longitud correspondiente. Con ello conseguimos en el disco el espacio deseado y ocupamos el número necesario de bloques (ver capítulo 2). Los bloques ocupados son anotados en el directorio, 16 Kbytes cada vez por entrada. Al abrir ahora nuestro fichero relativo se determina mediante el comando IINIT,RL qué bloques son ocupados por el fichero. Se almacenan todos los números del bloque (un bloque son 2 sectores consecutivos). Esta tabla es el fundamento para la gestión de ficheros relativos ya que sin ella no se podría acceder libremente a los distintos records. Se almacena además, por separado, la longitud de record, pero cuidado: no se comprueba si esta longitud es, en efecto, una potencia de dos. Incluso si abre un fichero relativo con la misma longitud de sector que otro fichero relativo previamente abierto, al comando OPENIN ha de seguirle ineludiblemente el comando INIT, ya que en caso contrario se hallarian todavía en memoria los números de bloque antiguos.

Partiendo del número de bloque almacenado resulta siempre posible calcular el sector referido correspondiente, y ello se hace en el comandoIRECORD. La rutina está programada de tal modo que un sector sólo es vuelto a cargar si resulta imprescindible. Puede ser que ya se halle en memoria; este caso se da en especial cuando usted "lee por encima" todos los records del primero al último, con el consiguiente

ahorro de tiempo. Además ha de determinarse, antes de leer un sector, si el sector que se halla actualmente en memoria fue escrito mediante IPUT. En caso afirmativo, se almacena el sector antiguo antes de cargar el nuevo.

Con la orden IPUT se copian en el buffer de sector todas las variables de cadena dadas, en la IGET se desvía simplemente el indicador de las variables de cadena hacia el buffer de sector; su longitud es igualmente modificada.

Estas son las líneas generales del principio hecho realidad en su rutina. Nos hemos decidido por los comandos RSX porque son más sencillos de manejar y mucho más potentes a la vez que algunos CALLs. Además, usted no ha de pensar (casi) en qué zona de memoria se halla la rutina.

Si no desea teclear los listados, sepa que existe un disco complementario al presente libro, que contiene todos los programas descritos.

Listado Assembler para los comandos de ampliación:

```
ORG      #8000
;
;
;Gestión de ficheros relativos con ordenadores Amstrad
;*****  

;Comandos RSX incorporados:
;  

;AN
;AUS
;ERR
;RECORD
;GET
;PUT
;INIT
;  

;JS 12/2/1986  

;  

LD      A, (VER)      ;EVITAR CAIDA ANTE
OR      A              ;DOBLE IAN
RET    NZ
CALL   #B900          ;ACTIVAR ROM
PUSH   AF
LD      A, (#DE01)    ;GUARDAR
LD      (VER), A       ;VERSION
POP    AF
CALL   #B90C          ;RESTORE
LD      A, (VER)
CP      #71
JR      Z, RSXON      ;464
LD      A, #DF
LD      (A1), A
LD      (A2), A
LD      A, (VER)
CP      #C9            ;6128?
JR      NZ, M02        ;664 ORDENADOR
```

```
LD      HL,TABU4
JR      M02+3.
M02:LD  HL,TABU2
LD      (A1+1),HL
LD      (A2+1),HL
RSXON:LD BC,RSX      ;AMPLIACIONES
LD      HL,KERNEL    ;4 BYTES DE RAM PARA
JP      #BCD1        ;UNIR AMPLIACIONES
|
RSX:DEFW TABLE       ;DIRECCION DE PALABRAS DE ORDEN
JP      RECORD
JP      GET
JP      PUT
JP      INIT         ;DETERMINAR LONGITUD
JP      SET          ;CONECTAR
JP      RESET        ;DESCONECTAR
JP      GETTXT       ;RECOGE TEXTO
|
TABLE:DEFM "RECORD"
DEFB    "D" + #80
DEFB    "G", "E", "T"+#80
DEFB    "P", "U", "T"+#80
DEFB    "I", "N", "I", "T"+#80
DEFB    "A", "N"+#80
DEFB    "A", "U", "S"+#80
DEFB    "E", "R", "R"+#80
DEFB    O             ;FIN DE TABLA
|
KERNEL:DEFS 4        ;MEMORIA PARA KERNEL
|
RECORD:CP 01          ;1. PARAMETRO
RET    NZ              ;CANTIDAD ERRONEA
LD      B,(IX+1)      ;RECOGE RECORD#
LD      C,(IX)
LD      HL,(RECLEN)   ;LONGITUD RECORD
A0:CALL  MULTI        ;DE=HL*BC
IX      DE,HL
LD      DE,512         ;LONGITUD DE UN SECTOR
A1:CALL  #BCD1        ;HL=HL/DE -- DE=RESTO
LD      A,(FLO)
```

```

LD      (OFFSET)DE ; GUARDAR OFFSET
LD      DE, 128    ; SUMAR 128 A CARRY
OR      A          ; SI EL FLAG Z=0 NO SUPONE TRANSFERENCIA
JR      Z, NOC
ADD    HL, DE
NOC: LD   (SECNR), HL ; GUARDAR NUMERO DE SECTOR
XOR    A; A:=0
LD      (POINTER), A ; PUNTERO A CERO
LD      (POINTER+1), A
LD      DE, (OLDSEC)
SBC    HL, DE      ; MISMO SECTOR?
RET    Z          ; SI CERO A BUFFER
CALL   SAVE        ; PREVIAMENTE, ALMACENAR BUFFER
LD      HL, (SECNR) ; RECOGE NUMERO DE SECTOR
LD      (OLDSEC), HL ; GUARDAR SECTOR
RR      H; /2
RR      L; /2
PUSH   AF          ; GUARDAR CARRY
LD      DE, (#A79B) ; TABLA DE BLOQUE
ADD    HL, DE      ; RECOGE NUMERO DE BLOQUE
LD      A, (HL)    ; NUMERO DE BLOQUE
LD      L, A
LD      H, 0        ; HL:=NUMERO DE BLOQUE
ADD    HL, HL      ; *2
POP    AF          ; RECUPERA CARRY
LD      DE, 0
ADC    HL, DE      ; SUMA CARRY
LD      DE, 9
A2: CALL #BDC1    ; HL=HL/DE -- DE:=RESTO
INC    HL          ; +1
INC    HL          ; +1 = TRACK
LD      A, L        ; GUARDAR
LD      (TRACK), A ; TRACK
LD      A, E        ; SECTOR
ADD    A, #41       ; SUMAR OFFSET
LD      (SECTOR), A ; GUARDAR SECTOR
LD      D, L        ; D:=TRACK
LD      C, A        ; C:=SECTOR
LD      A, (#A708)  ; DRIVE PARA OPENIN
LD      E, A

```

```
LD      HL, (#A751) ;PUNTERO PARA BUFFER DE ENTRADA
DEFB    #DF
DEFW    TAB1         ;CALL #C666, CARGA SECTOR
RET
;
;ALMACENAMIENTO DEL BLOQUE
;
SAVE: LD   A, (REAWRI) ;FLAG DE LECTURA/ESCRITURA
OR    A
RET   Z               ;ESCRITURA NO PRECISA
LD   HL, (TRACK)    ;TOMAR TRACK/SECTOR
LD   D, L            ;D:=TRACK
LD   C, H            ;C:=SECTOR
LD   A, (#A708)      ;DRIVE EN OPENIN
LD   E, A            ;E:=DRIVE
LD   HL, (#A751)    ;BUFFER DE ENTRADA
DEFB    #DF
DEFW    TAB2         ;CALL #C64E, ESCRIBIR SECTOR
XOR   A              ;ACCU:=0
LD   (REAWRI), A    ;REPONER FLAG
RET
;
GET: LD   C, 0        ;0=GET
JR    GETPUT
PUT: LD   C, 1        ;1=PUT
GETPUT: OR  A          ;COMPRUEBA CANTIDAD DE PARAMETROS
RET   Z               ;=0 => ENTONCES FIN
LD   B, A            ;CANTIDAD DE VARIABLES (CADENAS!)
ADD  A, A            ;ACCU*2
PUSH IX             ;IX A
POP   HL              ;HL
ADD  A, L
LD   L, A
LD   A, H
ADC  A, 0            ;HL + CANTIDAD*2-1 =PUNTERO EN PARAMETRO
LD   H, A
DEC   HL
LD   A, C            ;ORDER#
LD   (ORDER), A      ;GUARDAR
L0:PUSH HL           ;GUARDAR
```

```
LD      HL, (#A751)    ;PUNTERO PARA OPENIN
LD      DE, (OFFSET)
ADD    HL,DE           ;SUMAR OFFSET
LD      DE,(POINTER)
ADD    HL,DE           ;SUMAR PUNTERO
EX      DE,HL           ;DE:=PUNTERO A BUFFER
POP    HL              ;RECOGER DIRECCION
PUSH   BC
LD      B, (HL)
DEC    HL
LD      C, (HL)         ;DIRECCION VARIABLE EN BC
DEC    HL
PUSH   HL
PUSH   DE
PUSH   BC
LD      A, (ORDER)
OR     A
JR      NZ,PUTV        ;REALIZAR PUT A LAS VARIABLES
LD      B,0              ;CONTADOR A CERO
L1:LD  A, (DE)          ;RECUPERA CARACTER
CP      13               ;CR=FIN?
JR      Z,END
INC    B                ;INCREMENTAR LONGITUD
JR      Z,END            ;ERROR -- LONG.EXC. --> CADENA VACIA!
INC    DE
JR      L1
END:POP HL
LD      (HL),B           ;GUARDAR LONGITUD
POP    DE              ;DE:=DIRECCION DE INICIO
INC    HL
LD      (HL),E           ;BYTE BAJO
INC    HL
LD      (HL),D           ;BYTE ALTO
LD      HL,(POINTER)
LD      D,0
LD      E,B              ;E:=LONGITUD
INC    DE              ;+1 PARA CR
ADD    HL,DE
LD      (POINTER),HL    ;NUEVO PUNTERO
LOOP:POP HL              ;PUNTERO A TABLA DE PARAMETROS
```

```
POP      BC
DJNZ    LO      ;SIGUIENTE VARIABLE
RET
;
PUTV: POP  HL      ;RECOGE DIRECCION VARIABLE
LD      C, (HL)   ;CANTIDAD DE CARACTERES
LD      B, 0
INC    HL
LD      E, (HL)   ;BYTE BAJO DE DIRECCION
INC    HL
LD      D, (HL)   ;Y BYTE ALTO
EX      DE, HL    ;HL:=CADENA
POP    DE      ;RECUPERA DIRECCION BUFFER
LD      A, C
OR      A
PUSH   BC      ;GUARDA CANTIDAD
JR      Z, EN1    ;FIN ANTE CADENA VACIA
LDIR
EN1: EX  DE, HL  ;DE=SIMBOLO SEPARADOR
LD      (HL), 13
POP    DE
INC    DE      ;+1 PARA CARACTER SEPARADOR
LD      HL, (POINTER)
ADD   HL, DE
LD      (POINTER), HL ;NUEVO PUNTERO
LD      A, 1
LD      (REAWRI), A ;GUARDAR ESCRITO
JR      LOOP    ;CERRAR BUCLE
;
;INICIALIZA FICHERO
;
INIT: CP  01      ;1 PARAMETRO ?
RET    NZ      ;CANTIDAD ERRONEA DE PARAMETROS
LD      H, (IX+1)
LD      L, (IX)   ;DETERMINAR LONGITUD
LD      A, H
OR      'L
JP      Z, SAVE   ;COMANDO INIT, 0
LD      (RECLEN), HL ;ANOTAR
LD      HL, #FFFF
```

```
LD      (OLDDSEC),HL      ; BORRAR FLAG
LD      DE,(#A79B)        ; DIRECCION BUFFER OPENOUT
L10:LD  HL,#A719        ; TABLA DE BLOQUES
LD      B,16
L11:LD  A,(HL)
OR      A
RET    Z                  ; ULTIMO BLOQUE
LD      (DE),A
INC    DE
INC    HL
DJNZ   L11                ; BLOQUE SIGUIENTE
LD      HL,(#A729)
LD      BC,#80
ADD   HL,BC
LD      (#A729),HL
LD      HL,0000
LD      (#A768),HL      ; VACIAR BUFFER
DEFB   #DF
DEFW   GETCHAR          ; CALL #CF64 DISC IN CHAR
JR    L10
;
RECLEN:DEFW 64            ; LONGITUD DE RECORD 1 - 512
SECNR:DEFW 0              ; NUMERO DE SECTOR
OFFSET:DEFW 0             ; OFFSET EN EL SECTOR
REAWRI:DEFB 0             ; FLAG 1=ESCRITO, 0=LEIDO
OLDSEC:DEFW #FFFF         ; ULTIMO SECTOR
TRACK:DEFB 0              ; PISTA DEL SECTOR
SECTOR:DEFB 0             ; SECTOR DEL SECTOR
TAB1:DEFB #66,#C6,7       ; DIRECCION #C666 EN ROM DEL FLOPPY
TAB2:DEFB #4E,#C6,7       ; ESCRIBIR SECTOR
POINTER:DEFW 0            ; PUNTERO A BUFFER
ORDER:DEFB 0
GETCHAR:DEFB #64,#C7,7    ; CF64 GET CHAR DE BUFFER DE OPENIN
FLO:DEFB 0
FL1:DEFB 0
ANAUS:DEFB 0
;
MULTI:XOR A              ; BORRAR ACUMULADOR
LD      D,A              ; DE ES EL REGISTRO DE RESULTADO
LD      E,A
```

```
LD      (FLO),A
LD      (FL1),A
LD      A,16          ;CONTADOR
Q1:RR  B             ;SHIFT DE BC
RR      C
JR      NC,Q2
PUSH   HL
ADD    HL,DE
JR      NC,Q0          ;NINGUNA TRANSFERENCIA
LD      (FLO),A          ;ANOTAR CARRY
Q0:EX  DE,HL
PUSH   AF
LD      A,(FLO)
LD      HL,FL1
OR      (HL)
LD      (FLO),A          ;ANOTAR CARRY
POP    AF
POP    HL
Q2:ADD HL,HL
JR      NC,Q3
LD      (FL1),A          ;CONTADOR
Q3:DEC A              ;CONTADOR
JR      NZ,Q1
RET
;
;
;RUTINA PARA LA INTERCEPCION DE ERRORES
;*****
;
;(C) 1985 BY DATA BECKER GMBH
;
; JS 12/2/1986
;
;
;
SET:LD  A,(ANAUS)
OR      A              ;EVITA ENCENDIDO DOBLE
RET    NZ
LD      A,(VER)
LD      (ANAUS),A
```

```
CP      #71;464?
JR      NZ,NO1
LD      A,#C3
LD      (#AC01),A      ;READY
NO1:LD  A,#C3
LD      (#BB5A),A      ;OUT CHAR
LD      (#BB06),A      ;KW WAIT CHAR
LD      (#BC9B),A      ;CAS CATALOG
;
LD      A,(VER)        ;VERSION
CP      #71
JR      NZ,NO3          ;664
LD      HL,READYMODE
LD      (#AC02),HL      ;PATCH A READY
NO3:LD  HL,(#BB5B)
LD      (VEK4+1)<HL
LD      HL,OUTCHK        ;PATCH A OUT CHAR
LD      (#BB5B),HL      ;PATCH A OUT CHAR
LD      HL,(#BB07)        ;GUARDAR ANTIGUO VECTOR
LD      (VEK5+1),HL
LD      HL,KWC
LD      (#BB07),HL      ;PATCH A KW WAIT CHAR
LD      HL,(#BC9C)        ;ANTIGUO VECTOR
LD      (VEK3),HL
LD      HL,CAT
LD      (#BC9C),HL      ;CAS CATALOG
LD      HL,BUFFER
LD      (HELP),HL
RET
;
;
;LA RUTINA COMPRUEBA SI VIENE CARACTER DEL FLOPPY
;
OUTCHK:EX  (SP),HL      ;RECOGE DIRECCION DE RETORNO DE SALTO
PUSH    AF                ;RESCATA ACUMULADOR
LD      A,H                ;COMPRUEBA BYTE ALTO
CP      #CB                ;PROVIENE DE LA ROM 7 DE FLOPPY ?
JR      NZ,NEIN            ;NO
LD      A,1
LD      (TESTERR),A        ;FIJAR FLAG
```

```
POP      AF          ;NO DAR SALIDA A CARACTER
PUSH     HL          ;RECUPERA HL
LD       HL, (HELP)  ;RECOGE POINTER DE BUFFER
CP       10          ;ES EL CARACTER LF
JR       Z,NOT
;CR SE ADMITE COMO SIMBOLO SEPARADOR
LD       (HL),A      ;CARACTER EN MEMORIA
INC     HL
LD       A,L
CP       40+BUFFER&#FF   ;40 CARACTERES ?
JR       NZ,N1
DEC     HL          ;40 CARACTERES ES EL LIMITE SUPERIOR
N1:LD   (HELP),HL
NOT:POP  HL
PUSH    AF
SUPRESS:POP  AF
EX      (SP),HL      ;DIRECCION DE RETORNO DE SALTO
RET    ;NINGUNA SALIDA
;
;
NO:LD   A,(VER)
CP      #71
JR      Z,NEINALT
LD      A,(TESTERR)
OR      A
JR      Z,N12
DEC     A
N12:LD  (TESTERR),A
JR      NZ,SUPRESS
PUSH   HL
LD      HL,BUFFER
LD      (HELP),HL
POP    HL
POP    AF          ;RECOGE CARACTER
EX      (SP),HL
VEK4:DEFB #C7,0,#94 ;RST 2
;
NEINALT:LD  A,(TESTERR)
OR      A
JR      NZ,SUPRESS
```

```
POP      AF          ; RECOGE CARACTER
EX      (SP),HL
DEFB    #CF,0,#94 ; RST 2
;
;
; RUTINA PARA MODO READY
;
; READYMODE: LD A, (TESTERR)
OR      A          ; FIJA FLAGS
RET    Z          ; NINGUN ERROR
XOR    A
LD      (TESTERR),A
LD      HL,BUFFER
LD      (HELP),HL
LD      E,18
JP      #CA94
;
; PATCH PARA #BB06 KM WAIT CHAR
; VALOR DE DEFECTO PARA ERRORES DE HARDWARE
; ES C=CANCEL
;
KWC:PUSH  AF          ; RECUPERA ACUMULADOR
LD      A,(TESTERR)
OR      A          ; FIJA FLAGS
JR      NZ,DEFAULT
POP    AF          ; NINGUN C DE DEFECTO
VEK5:DEFB  #CF,#3C,#9A
DEFAULT:LD A,4
LD      (TESTERR),A
POP    AF
LD      A,"C"        ; DEFECTO
RET
;
;
;
;
; RETORNO DE RUTINA POR ERROR DE TEXTO
;
GETTXT:CP  01          ; 1 PARAMETRO ?
```

```
RET      NZ          ; CANTIDAD ERRONEA DE PARAMETROS
LD       E,(IX)     ; BYTE BAJO
LD       D,(IX+1)   ; BYTE ALTO
LD       C,#FF      ; CONTADOR
LD       HL,BUFFER
LA:LD   A,(HL)
CP       13         ; CARRIAGE RETURN?
INC      HL
JR       Z,LA
DEC      HL         ; HALLADO PRIMER CARACTER DE TEXO?
PUSH    HL         ; GUARDAR
DEC      HL         ; MENOS UNO
SLOOP: INC C
INC      HL
LD       A,(HL)
CP       13         ; CARRIAGE RETURN?
JR       NZ,SLOOP
LD       A,C
LD       (DE),A      ; ANOTAR LONGITUD
POP      HL         ; DIRECCION TEXTO
EX       DE,HL
INC      HL
LD       (HL),E      ; BYTE BAJO
INC      HL
LD       (HL),D      ; BYTE ALTO
RET
;
;
;REINICIALIZACION DEL PUNTERO
;
RESET: LD  A,(VER)    ; NUMERO DE VERSION
CP      #71
JR      NZ,N02
LD      A,#C9
LD      (#AC01),A
N02: LD  A,#CF      ; RST 2
LD      (#BB5A),A    ; OUT CHAR
LD      (#BB06),A    ; KM WAIT CHAR
LD      A,#DF      ; RST 3BH
LD      (#BC9B),A    ; CAS CATALOG
```

```
;  
LD      HL, (VEK4+1)    ;OUT CHAR  
LD      (#BB5B), HL  
LD      HL, (VEK5+1)    ;KM WAIT CHAR  
LD      (#BB07), HL  
LD      HL, #A88B  
LD      (#BC9C), HL    ;CAS CATALOG  
XOR    A  
LD      (ANAUS), A  
RET  
;  
CAT:PUSH AF             ;RECUPERA REGISTRO  
PUSH   HL  
CALL   RESET            ;DEVUELVE VALOR ORIGINAL  
POP    HL  
POP    AF  
CALL   #BC9B            ;CAS CATALOG  
PUSH   HL  
PUSH   AF  
CALL   SET              ;Y VOLVER A CONECTAR  
POP    AF  
POP    HL               ;DEVUELVE REGISTRO  
RET  
;  
;PUNTEROS Y MEMORIA AUXILIAR  
;*****  
;  
VEK3:DEFW 0              ;VECTOR OUTCHAR  
TESTERR:DEFB #0  
VER:DEFB 0  
HELP:DEFW BUFFER  
TABU2:DEFB #B0, #DD, #FD  
TABU4:DEFB #AE, #DD, #FD  
BUFFER:DEFM "OK"  
DEFB    #0D ;CR  
DEFS    40-3           ;BUFFER DE 40 CARACTERES
```

```
10'*****
20'*** Cargador BASIC para comandos RSX *****
30'*** JS 12/2/1986 (c) by DATA BECKER GmbH ***
40'*****
50'
60 DEFINT b-z : DEFREAL s,i
70 adresse = &6000 'Direccion inicial de la rutina
80'
90 DATA 3A,3F,83,B7,C0,CD,00,B9,F5,3A,01,DE,32,3F,83,F1
100 DATA CD,0C,B9,3A,3F,83,FE,71,28,1D,3E,DF,32,87,80,32
110 DATA CA,80,3A,3F,83,FE,C9,20,05,21,45,83,18,03,21,42
120 DATA 83,22,88,80,22,CB,80,01,40,80,21,70,80,C3,D1,BC
130 DATA 57,80,C3,74,80,C3,FF,80,C3,03,81,C3,7D,81,C3,04
140 DATA 82,C3,FA,82,C3,D3,82,52,45,43,4F,52,C4,47,45,D4
150 DATA 50,55,D4,49,4E,49,D4,41,CE,41,55,D3,45,52,D2,00
160 DATA 00,00,00,00,FE,01,C0,DD,46,01,DD,4E,00,2A,BA,81
170 DATA CD,D4,81,EB,11,00,02,CD,C1,BD,3A,D1,81,ED,53,BE
180 DATA 81,11,80,00,B7,28,01,19,22,BC,81,AF,32,CB,81,32
190 DATA CC,81,ED,5B,C1,81,ED,52,CB,CD,E6,80,2A,BC,81,22
200 DATA C1,81,CB,1C,CB,1D,F5,ED,5B,9B,A7,19,7E,6F,26,00
210 DATA 29,F1,11,00,00,ED,5A,11,09,00,CD,C1,BD,23,23,7D
220 DATA 32,C3,81,7B,C6,41,32,C4,81,55,4F,3A,08,A7,5F,2A
230 DATA 51,A7,DF,C5,81,C9,3A,C0,81,B7,C8,2A,C3,81,55,4C
240 DATA 3A,08,A7,5F,2A,51,A7,DF,C8,81,AF,32,C0,81,C9,0E
250 DATA 00,18,02,0E,01,B7,C8,47,87,DD,E5,E1,85,6F,7C,CE
260 DATA 00,67,2B,79,32,CD,81,E5,2A,51,A7,ED,5B,BE,81,19
270 DATA ED,5B,CB,81,19,EB,E1,C5,46,2B,4E,2B,E5,D5,C5,3A
280 DATA CD,81,B7,20,24,06,00,1A,FE,0D,28,06,04,28,03,13
290 DATA 18,F5,E1,70,D1,23,73,23,72,2A,CB,81,16,00,58,13
300 DATA 19,22,CB,81,E1,C1,10,BF,C9,E1,4E,06,00,23,5E,23
310 DATA 56,EB,D1,79,B7,C5,28,02,ED,B0,EB,36,0D,D1,13,2A
320 DATA CB,81,19,22,CB,81,3E,01,32,C0,81,18,D7,FE,01,CO
330 DATA DD,66,01,DD,6E,00,7C,B5,CA,E6,80,22,BA,81,21,FF
340 DATA FF,22,C1,81,ED,5B,9B,A7,21,19,A7,06,10,7E,B7,C8
350 DATA 12,13,23,10,FB,2A,29,A7,01,80,00,09,22,29,A7,21
360 DATA 00,00,22,68,A7,DF,CE,81,18,DE,40,00,00,00,00,00
370 DATA 00,FF,FF,00,00,66,C6,07,4E,C6,07,00,00,00,64,CF
```

```
380 DATA 07,00,00,00,AF,57,5F,32,D1,81,32,D2,81,3E,10,CB
390 DATA 18,CB,19,30,15,E5,19,30,03,32,D1,81,EB,F5,3A,D1
400 DATA 81,21,D2,81,B6,32,D1,81,F1,E1,29,30,03,32,D2,81
410 DATA 3D,20,DC,C9,3A,D3,81,B7,CO,3A,3F,83,32,D3,81,FE
420 DATA 71,20,05,3E,C3,32,01,AC,3E,C3,32,5A,BB,32,06,BB
430 DATA 32,9B,BC,3A,3F,83,FE,71,20,06,21,AB,82,22,02,AC
440 DATA 2A,5B,BB,22,9E,82,21,5B,82,22,5B,BB,2A,07,BB,22
450 DATA C8,82,21,BF,82,22,07,BB,2A,9C,BC,22,3C,83,21,2A
460 DATA 83,22,9C,BC,21,48,83,22,40,83,C9,E3,F5,7C,FE,CB
470 DATA 20,1E,3E,01,32,3E,83,F1,E5,2A,40,83,FE,0A,28,0B
480 DATA 77,23,7D,FE,70,20,01,2B,22,40,83,E1,F5,F1,E3,C9
490 DATA 3A,3F,83,FE,71,28,19,3A,3E,83,B7,28,01,3D,32,3E
500 DATA 83,20,EA,E5,21,48,83,22,40,83,E1,F1,E3,CF,00,94
510 DATA 3A,3E,83,B7,20,D7,F1,E3,CF,00,94,3A,3E,83,B7,C8
520 DATA AF,32,3E,83,21,48,83,22,40,83,1E,12,C3,94,CA,F5
530 DATA 3A,3E,83,B7,20,04,F1,CF,3C,9A,3E,04,32,3E,83,F1
540 DATA 3E,43,C9,FE,01,CO,DD,5E,00,DD,56,01,0E,FF,21,48
550 DATA 83,7E,FE,0D,23,28,FA,2B,E5,2B,0C,23,7E,FE,0D,20
560 DATA F9,79,12,E1,EB,23,73,23,72,C9,3A,3F,83,FE,71,20
570 DATA 05,3E,C9,32,01,AC,3E,CF,32,5A,BB,32,06,BB,3E,DF
580 DATA 32,9B,BC,2A,9E,82,22,5B,BB,2A,C8,82,22,07,BB,21
590 DATA BB,A8,22,9C,BC,AF,32,D3,81,C9,F5,E5,CD,FA,B2,E1
600 DATA F1,CD,9B,BC,E5,F5,CD,04,82,F1,E1,C9,00,00,00,00
610 DATA 48,83,B0,DD,FD,AE,DD,FD,4F,4B,0D
620 :
630 FOR i=0 TO &34A
640   READ d$
650   POKE i+adresse, VAL("&" + D$)
660   s=s+VAL("&" + D$)
670 NEXT
680 IF s<>95703 THEN PRINT CHR$(7) "*** Error en Datas !! ***
": END
690 :
700 'Ajuste a la zona de memoria
710 :
720 FOR i=adresse TO adresse+&34A
730   p=PEEK(i)
740   IF p>&7F AND p<&84 THEN 800
750 NEXT
760 PRINT CHR$(7) : INPUT "Nombre del fichero :",a$
770 SAVE a$,b,adresse,&34B
```

```
770 SAVE a$,b,adresse,&348
780 END
790 :
800 IF PEEK(I+1)>&7F AND PEEK(I+1)<&84 OR PEEK(i-1)=i OR PEE
K(i-1)=17 THEN 750
810 ad=PEEK(i-1) + p*256 : ad=UNT(ad) - &8000 + adresse
820 POKE i-1,&FF AND UNT(ad)      'Byte bajo
830 POKE i,&FF AND INT(ad/256)   'Byte alto
840 GOTO 750
```

5.4 El programa de gestión de ficheros

¿Su colección de discos es un caos o no recuerda en qué clasificador se halla su sello más valioso? Pues el programa de gestión de ficheros le vendrá que ni pintado.

El programa de gestión de ficheros es relativamente largo, pero no debería abstenerse de teclearlo (o de adquirir el correspondiente disco del libro), ya que le abre posibilidades insospechadas. No es tan cómodo como un DATAMAT u otros conocidos programas de gestión de ficheros, pero es suficiente, por velocidad y flexibilidad, para muchísimas aplicaciones.

Da lo mismo si desea gestionar su agenda o su discoteca, ya que no está prevista ninguna máscara de entradas fija. Dispone usted de hasta 10 campos de datos para sus juegos de datos, que puede ocupar a su gusto. El programa está estructurado de modo que pueda gestionar hasta 200 juegos de datos. Los datos se almacenan secuencialmente, no de modo relativo; todos los datos se recogen en memoria. De no haberse hecho así, el programa hubiera resultado más largo aún.

Se incorporaron a este programa los caracteres especiales españoles, si bien éstos no resultan correctamente ordenados en el proceso de ordenación (!), ya que la tarea hubiera resultado desproporcionada frente a los resultados.

Resulta además aconsejable que en el disco en el que almacene el programa de gestión de ficheros se halle también la rutina de intercepción de errores y/o el paquete de ampliación del capítulo 5.3 (bajo los correspondientes nombres) de fichero). Genérelo, consecuentemente con el

listado del capítulo 5.3, con la dirección de inicio &9F00 y almacénelo bajo el nombre "RSX.BIN". Eso es todo lo que ha de pensar.

Tras teclear el programa, haberlo almacenado al menos una vez (mejor varias veces en distintos discos) y de que exista el fichero RSX.BIN, puede iniciar el programa. Obtendrá en primer lugar el menú en pantalla. Mediante las teclas cursoras puede elegir los puntos del menú, confirmándolos mediante ENTER; se ejecuta entonces el punto de menú elegido.

Los datos se almacenan bajo user 1, de modo que los datos queden claramente separados de los programas y datos normales. Para ver sus ficheros puede elegir el punto del menú

'Mostrar directorio'

, el cual le mostrará únicamente los ficheros bajo user 1, es decir, sus propios ficheros. Cuando lo ejecute por primera vez, comprobará que en la pantalla se muestra solamente la memoria disponible de su disco. Pulsando una tecla cualquiera se vuelve al menú principal.

Volvamos a considerar los datos que ya tomamos como ejemplo en los capítulos 1 y 5.3. Creamos un FICHERO TELEFONICO con los tres campos de datos siguientes:

Apellido
Nombre
Teléfono

Podemos ampliar facilmente el fichero en tres campos más incluyendo, por ejemplo, los campos de datos:

Provincia
Población
Calle

Elija el punto de menú

'Crear/Variar máscara'

, y crearemos nuestra máscara o plantilla con los seis campos de datos. Las entradas se realizan como siempre mediante ENTER. Si desea modificar un campo, puede hacer subir el cursor mediante la tecla correspondiente, y sobreescribir el campo que proceda. Por motivos internos del sistema operativo no resulta posible corregir letras individuales, de modo que vuelve a llenar completamente el campo correspondiente. Tras una entrada, el cursor salta al campo siguiente. Si desea abandonar el punto del menú

'Crear/Variar máscara'

pulse simultáneamente las teclas CTRL y ENTER. Retornará instantáneamente al menú principal. Tan pronto haya entrado un juego de datos, puede corregir la máscara, pero no ampliar el número de campos de datos.

Tras haber creado la máscara, debería elegir el punto del menú 'entrar/variар datos'. Obtendrá la máscara de entrada en el modo de 80 caracteres. Se le indica en la parte superior derecha la longitud máxima de sus campos de datos para evitar que una línea resulte sobreescrita.

El campo que està siendo ocupado se muestra invertido. Introduzca ahora sus propios datos. Cuando haya acabado, la màscara de entrada volverà a ser vaciada y en la parte inferior izquierda verà que ha sido almacenado un juego de datos. Probablemente haya observado con estupor una informaciòn de estado en la parte inferior derecha de la pantalla. Usted determina con estricta rigidez que lo que no debe ser no puede ser - asi que ha de haber un estado mès. Se trata del estado de correcciòn. En la ventana central ve que la pulsaciòn simultanea de las teclas <Ctrl> y ENTER lleva a una variaciòn del estado. Intentelo:

En la esquina inferior derecha, bajo "Estado" ya no està la palabra "entrada", sino "correcciòn". Aparece, ademàs

**** rellene la màscara con concepto de bùsqueda ****

Ello tiene el siguiente significado: si desea modificar un juego de datos ha de comunicar al programa el juego de datos del que se trata. Para ello està a su disposiciòn toda la màscara de entrada - pero no se asuste, no ha de llenar toda la màscara con datos. Introduzca por ejemplo su nombre en el campo "nombre", cierre los otros campos mediante ENTER. Tan pronto haya llegado al limite inferior de la màscara de entrada, todos los campos restantes seràn llenados con sus datos, que puede ahora corregir. Aquí vale lo mismo que en el caso de la màscara: los campos que deseé corregir han de ser totalmente sobreescritos, los campos que no deseé corregir, ciérrelos simplemente mediante ENTER. Cuando haya cerrado con ENTER todos los campos, se hallará de nuevo en el modo de entrada. Pero en caso de que tenga en su fichero mès de una persona con su nombre, puede suceder que otro juego de datos sea mostrado para correcciòn, ya que únicamente, es tomado en cuenta el primer juego que responde a su descripcìon.

Imaginese que entre sus amistades cuente usted con dos de nombre PEDRO. Si quiere asegurarse de que le sea mostrado el PEDRO correcto, ha de darle al programa adicionalmente más información que la del nombre, como por ejemplo, el número de teléfono o bien el apellido.

Si introduce más de un campo de datos, se lleva a cabo una 'comparación y', es decir, han de darse todas las condiciones. Si pidiera la búsqueda de un juego de datos inexistente, obtendría el mensaje de error:

**** el juego de datos no existe ****

Todavía se halla en el modo de corrección y puede emprender un nuevo intento.

Saldrá en cualquier momento del modo de corrección pulsando CTRL/ENTER, retornando al modo de entrada (sin que se lleve a cabo la corrección).

Saldrá en cualquier momento del menú 'entrada/corrección' entrando FIN en un campo de datos cualquiera.

Brevemente sobre los restantes puntos del menú:

Cargar y Almacenar datos

Se cargan/almacenan los datos entrados y la máscara. Puede incorporar un nombre de fichero de una longitud de hasta ocho caracteres. Retornará al menú principal con CTRL/ENTER.

Borrar datos

Puede usted borrar un juego de datos. Para ello puede buscarlo previamente al igual que en la parte de programa crear/variante; antes de que resulte borrado, ha de confirmar usted una pregunta de seguridad.

Ordenar datos

Puede ordenar sus datos tras cualquier campo, cargando previamente dicho campo.

Listar datos

Puede elegir si la salida ha de ser por impresora (I) o a pantalla (P). Tras pulsar la tecla "I" o bien "P" puede elegir los campos de datos a los que hay que dar salida; interesante para listas que no contengan todos los campos de datos. Para evitar la salida de un campo de datos, pulse en el campo correspondiente la tecla ENTER, o llene el campo con un carácter cualquiera; basta, por tanto, pulsar en un campo la tecla de espacio/ENTER para darle salida.

Mostrar directorio

Le serà mostrado en pantalla el índice del contenido del disco.

Finalizar programa

Elija este punto del programa únicamente cuando haya almacenado en disco sus datos y, eventualmente, sus correcciones; no hay pregunta de seguridad alguna en el programa.

Borrar todo

Se borran todos los datos en memoria, el programa comienza de nuevo.

```
10 ****
20 ***** (c) 1986 by DATA BECKER ****
30 ***** Autor : Joerg Schieb ****
40 ***** Gestion de ficheros ****
50 ****
60 *
70 ON ERROR GOTO 3170
80 SYMBOL AFTER 91
90 MEMORY &9E50-1
100 LOAD "0:RSX.BIN"
110 CLEAR
120 DEFINT A-Z : ds$="" : CALL &9E50 !AN
130 ON ERROR GOTO 3170
140 ON BREAK GOSUB 960
150 OPENOUT "dumm"
160 MEMORY HIMEM-1
170 CLOSEOUT
180 DIM ma$(20),m$(9),l(20),d$(200,9),da$(9)
190 cursorup$=CHR$(240):cursordown$=CHR$(241)
200 !USER,1
210 :
220 REM =====
230 REM      Definicion de los caracteres espanoles
240 REM =====
250 :
260 DATA 135,91,136,92,137,93,132,123,133,124,134,125,129,126
270 FOR i=1 TO 7:READ a,b:KEY a,CHR$(b):NEXT
280 SYMBOL 123,14,0,120,8,122,204,118
290 SYMBOL 124,14,0,60,102,126,96,60
300 SYMBOL 92,14,22,0,56,24,24,60
310 SYMBOL 91,14,22,0,60,102,102,62
320 SYMBOL 125,14,22,0,102,102,102,62
330 SYMBOL 93,112,14,0,192,102,102,102
340 SYMBOL 126,24,0,24,24,24,24,24
350 KEY DEF 29,1,124,92
360 KEY DEF 28,1,123,91
370 KEY DEF 26,1,125,93,124
```

```
380 KEY DEF 24,1,126,ASC("#")
390 KEY DEF 19,1,58,42
400 KEY DEF 17,1,59,43,64
410 KEY DEF 18,0,13,13,140
420 KEY 140,"*ESC*"+CHR$(13)
450 :
460 DATA Cargar datos,Almacenar datos,Entrar/modificar datos,
Borrar datos",Ordenar datos,Listar datos,Mostrar directorio,C
ear/Modi
ficiar mascara,Finalizar programa,Borrar todo,Fin
470 :
480 i=0 : WHILE ma$(i)<>"Fin" : i=i+1
490 READ ma$(i) : l(i)=LEN(ma$(i))/2
500 WEND : anzmask = i-1
510 STAT=0:INK 0,0 : INK 1,13 : BORDER 0 : PAPER 0 : MODE 1 :
PEN 1 : PEN #1,0 : PAPER #1,1
520 WINDOW #1,1,40,1,3:CLS #1:LOCATE #1,16,2:PRINT #1,"M E N
"
530 FOR i=1 TO anzmask
540 LOCATE 20-1(i),5+i*2 : PRINT ma$(i)
550 NEXT
560 feld=1 : PEN #1,1
570 PAPER #1,1 : PEN #1,0
580 WINDOW #1,20-1(feld),20+1(feld),5+feld*2,5+feld*2 : PRINT
#1,ma$(feld)
590 d$=INKEY$:IF d$="" THEN 590
600 PAPER #1,0:PEN #1,1:IF d$=cursorup$ THEN PRINT #1,ma$(feld
):feld=feld-1:IF feld=0 THEN feld=anzmask:GOTO 570 ELSE 570
610 IF d$=cursordown$ THEN PRINT #1,ma$(feld):feld=feld+1:IF
feld>anzmask THEN feld=1:GOTO 570 ELSE 570
620 IF d$<>CHR$(13) THEN 590
630 ON feld GOSUB 1680,1240,2190,2690,2810,2940,1940,2020,180
,640:GOTO 510
640 RUN 120 'Borrar todo
650 :
660 REM =====
670 REM     Busca el array de cadena da$
680 REM =====
690 :
700 IF anzah1=0 THEN found=0 : RETURN
```

```
710 FOR i=1 TO anzahl
720   FOR i9=0 TO anzahlfelder-1
730     IF da$(i9)="" THEN 750
740     IF da$(i9)<>d$(i,i9) THEN 770
750   NEXT i9
760   found=i : RETURN
770 NEXT i
780 found=0 : RETURN
790 :
800 REM =====
810 REM      Ordena segun numero de campo feld
820 REM =====
830 :
840 IF anzahl<2 THEN RETURN
850 EVERY 30,0 GOSUB 2870
860 FOR i1=1 TO anzahl-1
870   ver=i1
880   FOR i2=i1+1 TO anzahl
890     IF d$(i2,feld)<d$(ver,feld) THEN ver=i2
900   NEXT
910   FOR i2=0 TO anzahlfelder-1
920     d$=d$(ver,i2) : d$(ver,i2)=d$(i1,i2) : d$(i1,i2)=d$
930   NEXT
940 NEXT
950 i=REMAIN(0)
960 RETURN
970 :
980 REM =====
990 REM  Comprueba la existencia de un nombre de fichero
1000 REM =====
1010 :
1020 nf=0 : ef=0 : OPENIN file$+".dat" : IF ef THEN 1020
1030 found=1-nf : '1=hallado, 0=no hallado
1040 CLOSEIN
1050 RETURN
1060 :
1070 REM =====
1080 REM      Borra la cadena numero Sn
1090 REM =====
1100 :
1110 IF anzahl=1 THEN anzahl=0 : RETURN
```

```
1120 FOR i=sn TO anzahl-1
1130   FOR i1=0 TO anzahlfelder-1
1140     d$(i,i1)=d$(i+1,i1)
1150   NEXT
1160 NEXT
1170 anzahl=anzahl-1
1180 RETURN
1190 :
1200 REM =====
1210 REM   Almacenar el fichero en disco
1220 REM =====
1230 :
1240 GOSUB 1490 : IF file$="*ESC*" THEN RETURN ELSE GOSUB 102
  : IF found=0 THEN 1310 ELSE WINDOW #1,1,40,25,25 : PAPER 2 :
INK 2,1
: CLS #1 : PRINT #1,CHR$(7)"Ha de ser sustituido (s,n) "
1250 EVERY 20,0 GOSUB 1290
1260 d$=INKEY$ : IF d$="" THEN 1260
1270 im=REMAIN(0):IF UPPER$(d$)<>"S" THEN 1240
1280 GOTO 1310
1290 LOCATE #1,30,1 : IF im=0 THEN PRINT#1,"?";:im=1 ELSE PRI
T#1,"";:im=0
1300 RETURN
1310 ef=0 : OPENOUT file$+".dat" : IF ef THEN 1310
1320 PRINT#9,anzahlfelder
1330 FOR i=0 TO anzahlfelder-1
1340 PRINT #9,m$(i)
1350 NEXT
1360 PRINT#9,anzahl
1370 FOR i=1 TO anzahl
1380   FOR i1=0 TO anzahlfelder-1
1390     PRINT#9,d$(i,i1)
1400   NEXT
1410 NEXT
1420 CLOSEOUT
1430 RETURN
1440 :
1450 REM =====
1460 REM   Lee nombre de fichero
1470 REM =====
1480 :
```

```
1490 MODE 1 : WINDOW #1,1,40,1,3 : PAPER #1,2 : INK 2,1 : CLS#
1 : LOCATE #1,7,2 : PRINT #1,"Introduzca el nombre del fichero
:"
1500 WINDOW #1,1,40,15,16 : CLS #1 : PRINT#1,TAB(10)"No utilice (.,:"CHR$(34)"
1510 PRINT#1,TAB(8)"Introduzca como maximo 8 caracteres
1520 WINDOW #1,1,40,6,10 : CLS #1
1530 LOCATE #1,5,3 : INK 3,3
1540 PRINT #1,"Nombre de fichero :";
1550 WINDOW #1,16,24,8,8 : PAPER #1,3 : CLS #1
1560 LINE INPUT #1,"",file$ : IF file$="*ESC*" THEN RETURN
1570 IF LEN(file$)>8 THEN PEN 3:LOCATE 8,16:PRINT CHR$(7)"Introduzca como maximo 8 caracteres":FOR i=1 TO 300:NEXT:PAPER #1,
2:GOTO 1
500
1580 c$=".,:"+CHR$(34)
1590 FOR i=1 TO LEN(c$):IF INSTR (file$,MID$(c$,i,1))=0 THEN N
EXT:GOTO 1610
1600 LOCATE 10,15 : PEN 3:PRINT CHR$(7)"No utilice (.,:"CHR$(3
4)":":FOR i=1 TO 300:NEXT:PAPER #1,2:GOTO 1500
1610 RETURN
1620 :
1630 :
1640 REM =====
1650 REM Lee fichero
1660 REM =====
1670 :
1680 GOSUB 1490 : IF file$="*ESC*" THEN RETURN ELSE GOSUB 1020
: IF found=0 THEN 1680
1690 ef=0 : OPENIN file$+".dat" : IF ef THEN 1690
1700 INPUT#9,anzahlfelder
1710 ERASE d$:DIM d$(200,anzahlfelder-1)
1720 FOR i=0 TO anzahlfelder-1
1730 INPUT#9,m$(i)
1740 NEXT
1750 INPUT#9,anzahl
1760 FOR i=1 TO anzahl
1770   FOR i1=0 TO anzahlfelder-1
1780     INPUT#9,d$(i,i1)
1790   NEXT i1
1800 NEXT
```

```
1810 CLOSEIN
1820 RETURN
1830 :
1840 REM =====
1850 REM     Finalizar programa
1860 REM =====
1870 :
1880 !USER,0 : CALL aus : MODE 2 : PRINT "*** Programa finaliz
ado ***" : END
1890 :
1900 REM =====
1910 REM     Muestra contenido disco
1920 REM =====
1930 :
1940 MODE 2 : CAT
1950 LOCATE 36,25 : PRINT "<< Pulsacion de tecla >>"
1960 CALL &BB06 : RETURN
1970 :
1980 REM =====
1990 REM     Modifica/crea mascara
2000 REM =====
2010 :
2020 MODE 1 : WINDOW #1,1,40,1,3 : PAPER #1,2 : INK 2,1 : CLS#
1 : LOCATE #1,7,2 : PRINT #1,"Creacion y modificacion de la m
scara
2030 LOCATE 1,7 : IF anzahl>0 THEN bis=anzahlfelder-1 ELSE bis
=9
2040 FOR i=0 TO bis
2050   PRINT RIGHT$(STR$(i+1),2) ". Feld:"m$(i)
2060 NEXT
2070 WINDOW #1,5,35,23,25 : PAPER #1,2 : PEN #1,1 : CLS#1 : LO
CATE #1,8,2 : PRINT#1,"Ctrl/Enter f3r Ende
2080 WINDOW #1,10,40,7,7+bis : PAPER #1,0 : PEN #1,1
2090 CLS#1:FOR I=0 TO ANZAHLFELDER-1:PRINT #1,M$(I)
2100 NEXT:LOCATE #1,1,anzahlfelder+1+(anzahlfelder=bis+1):i2=V
POS(#1)-2:LINE INPUT #1,"",a$ : IF a$="*ESC*" THEN RETURN EL8#
IF a$="
" THEN LOCATE #1,1,anzahlfelder+1 : GOTO 2090
2110 i1=VPOS(#1)-2:IF i2=8 AND i1=8 THEN i1=9
2120 m$(i1)=a$ : IF i1>anzahlfelder-1 THEN anzahlfelder=i1+1
```

```
2130 GOTO 2090
2140 :
2150 REM =====
2160 REM Entrar/modificar datos
2170 REM =====
2180 :
2190 IF anzahlfelder=0 THEN RETURN ELSE MODE 2 : WINDOW #1,1,8
0,1,3 : PAPER #1,1 : PEN #1,0 : CLS #1
2200 IF stat=0 THEN a$="Entrar y modificar datos" ELSE IF stat
=3 THEN a$="Seleccion del juego de datos a inspeccionar" ELSE
IF stat=
5 THEN a$="Seleccion de los juegos de datos a emitir"
2210 LOCATE #1,40-LEN(a$)/2,2:PRINT #1,a$
2220 GOSUB 2580
2230 LOCATE 1,7 : FOR i=0 TO anzahlfelder-1
2240 PRINT m$(i)TAB(laenge-2)":""
2250 NEXT
2260 LOCATE 60,5 : PRINT "Longitud de campo de datos="80-laeng
e .
2270 PEN #2,0 : PEN #3,1 : PAPER #2,1 : PAPER #3,0
2280 WINDOW #2,5,75,22,23 : CLS #2 : IF stat<5 THEN PRINT #2,T
AB(7)** Ctrl/Tecla ENTER=Modificar estado (Entrada/Correccio
n)**:IF
stat=0 THEN a$="** Introduzca 'Fin' en cualquier campo para f
inalizar **":GOTO 2300
2290 IF stat>>2 THEN a$="** Rellene la mascara con concepto bu
scado **" ELSE a$="** Introduzca las correcciones **"
2300 PRINT #2,TAB(35-LEN(a$)/2)a$
2310 WINDOW #1,1,80,25,25 : PAPER #1,1 : PEN #1,0
2320 CLS#1 : LOCATE #1,5,1 : PRINT #1,"Almacenado :";anzahl :
LOCATE #1,55,1 : PRINT #1,"Estado : "; : IF stat<>1 THEN PRINT
#1,"** E
ntrada **" ELSE PRINT#1,"** Correccion **"
2330 PO=0
2340 IF stat<>2 THEN WINDOW #3,laenge,79,7,16 : CLS#3
2350 WHILE PO<anzahlfelder : WINDOW#2,1,laenge-3,7+PO,7+po : W
INDOW #3,1,laenge-3,7+po,7+po : CLS#2 : PRINT #2,m$(po)
2360 WINDOW #4,laenge,79,7+po,7+po : LINE INPUT #4,a$ : a$=LEF
T$(a$,80-laenge)
```

```
2370 IF a$="*ESC*" THEN 2500 ELSE IF LOWER$(a$)="Fin" THEN RET
URN
2380 da$(po)=a$
2390 CLS#3:PRINT#3,m$(po):po=po+1
2400 WEND
2410 IF stat=0 THEN anzahl=anzahl+1+(anzahl=200):FOR i=0 TO an
zahlfelder-1:d$(anzahl,i)=da$(i):NEXT:GOTO 2320
2420 IF stat=2 THEN 2470
2430 IF stat=5 THEN RETURN
2440 GOSUB 700 : REM proporciona cadena buscada
2450 LOCATE 25,20 : IF found=0 THEN PRINT CHR$(7)"** el juego
de datos no existe **" ELSE PRINT SPACE$(30)
2460 IF found=0 THEN 2320 ELSE FOR i=0 TO anzahlfelder-1:LOCAT
E laenge,7+i:PRINT d$(found,i):NEXT:IF stat=3 THEN RETURN ELSE
stat=2:
GOTO 2280
2470 FOR i=0 TO anzahlfelder-1:IF da$(i)="" THEN 2490
2480 d$(found,i)=da$(i)
2490 NEXT:stat=1:GOTO 2500
2500 IF stat=5 THEN RETURN ELSE IF st>1 THEN 2280 ELSE stat=1-
stat:GOTO 2280
2510 END
2520 RETURN
2530 :
2540 REM =====
2550 REM Proporciona la mayor cadena
2560 REM =====
2570 :
2580 laenge=0
2590 FOR i=0 TO anzahlfelder-1
2600 IF LEN(m$(i))>laenge THEN laenge=LEN(m$(i))
2610 NEXT
2620 laenge=laenge+4
2630 RETURN
2640 :
2650 REM =====
2660 REM Inspeccionar juego de datos
2670 REM =====
2680 :
2690 IF anzahl=0 THEN RETURN ELSE stat=3 : st1=1 : GOSUB 2190
: IF LOWER$(a$)="Fin" THEN RETURN ELSE st1=0
2700 LOCATE 1,18 : PRINT "Juego de datos ok (s/n) ":EVERY 30,0
GOSUB 2740
```

```
2710 d$=INKEY$ : IF d$="" THEN 2710
2720 sn=REMAIN(0):IF LOWER$(d$)<>"s" THEN 2690
2730 sn=found : GOSUB 1110 : RETURN
2740 i=ABS(i=0):LOCATE 20,18:IF i THEN PRINT"??" ELSE PRINT" "
2750 RETURN
2760 :
2770 REM =====
2780 REM      Ordenar
2790 REM =====
2800 :
2810 MODE 2:PRINT"Segun que campo ha de ordenarse (1-""STR$(anz
ahlfelder)":"; : INPUT " ",a
2820 IF a<1 OR a>anzahlfelder THEN 2810
2830 PRINT : PRINT "Campo "CHR$(34);m$(a-1);CHR$(34)" -- ok ?
(s/n)
2840 d$=INKEY$ : IF d$="" THEN 2840
2850 IF LOWER$(d$)<>"s" THEN 2810
2860 feld=a-1 : GOTO 840
2870 i9=(i9=0):LOCATE 5,7:IF i9 THEN PRINT" "CHR$(224) ELSE PR
INT CHR$(224)" "
2880 RETURN
2890 :
2900 REM =====
2910 REM      Salida a pantalla e impresora
2920 REM =====
2930 :
2940 MODE 2:PRINT"Salida a impresora o pantalla ? (I/P)"
2950 d$=INKEY$ : IF d$="" THEN 2950
2960 IF INSTR("pi",LOWER$(d$))=0 THEN 2950
2970 IF LOWER$(d$)="i" THEN ae=8 ELSE ae=0
2980 st1=1:stat=5:GOSUB 2190
2990 f=0
3000 FOR i=0 TO anzhlfelder-1
3010   IF da$(i)<>"" THEN f(f)=i:f=f+1
3020 NEXT
3030 laenge=laenge-4:IF ae=0 THEN MODE 2
3040 FOR i=1 TO anzahl
3050   FOR i1=0 TO f-1
3060     PRINT#ae,m$(f(i1))SPACE$(laenge-LEN(m$(f(i1))))": "d$
(i,f(i1))
```

```
3070 NEXT
3080 PRINT#ae
3090 NEXT
3100 IF ae=0 THEN PRINT "<< Pulsacion de tecla >>":CALL &BB06
3110 RETURN
3120 :
3130 REM =====
3140 REM     Rutina de errores
3150 REM =====
3160 :
3170 IF ERR<>18 THEN RESUME NEXT
3180 ef=0 : CALL msg,@ds$
3190 nf=0:IF RIGHT$(ds$,9)="not found" THEN nf=1:RESUME NEXT
3200 WINDOW #6,1,40,24,25:PAPER #6,1: PEN #6,0:CLS #6
3210 PRINT #6,CHR$(7)"*** Disk:";ds$;" in";ERL:PRINT #6,"    <
ENTER>=Menu principal, en caso contrario, de nuevo
3220 d$=INKEY$ : IF d$="" THEN 3220
3230 ef=1 : IF ASC(d$)=13 THEN RESUME 510
3240 RESUME NEXT
```

5.5 DISCMON - monitor de discos para el CPC

¿Ya ha probado el programa para leer y mostrar sectores individuales del disco?. Pues ya habrá comprobado la lentitud de salida a pantalla en este programa. En el monitor de disco que ahora presentamos, basado inicialmente en el anterior programa, hemos incluido la salida de la información de sectores en un corto programa en lenguaje máquina, para alcanzar una velocidad de salida aceptable. Hemos incorporado también algunos comandos supplementarios, que serán de gran ayuda en una u otra ocasión. Son los siguientes:

- Leer sectores
- Modificar contenido del sector
- Escribir sectores
- Mostrar directorio
- Calcular números de pista y sector a partir del número de bloque

La lectura de los sectores está estructurada de una forma bastante cómoda. Con las cuatro teclas cursoras puede alcanzarse cualquier sector del disco. Para ello se cuenta con las teclas 'Cursor Up' y 'Cursor Down' para la elección de la pista siguiente o anterior, y con las teclas 'Cursor Left' y 'Cursor Right' para leer el sector anterior o posterior al actual.

En estas funciones hay un 'Wrap Around' incorporado. Así que si lee el sector 9 de una pista y pulsa tras ello la tecla 'Cursor Right', le será mostrado el sector 1 de la pista siguiente.

Como particularidad, manteniendo pulsadas las teclas de cursor no se muestra el contenido del sector recién leído. Ello acelera ostensiblemente el acceso. Lamentablemente no se muestra el último sector leído, de modo que debería volver a leerlo con 'R (ENTER)'.

Por supuesto que puede elegir también un sector concreto para lo que nos servimos del comando 'R'. Pero en vez de entrar ENTER, indique un número de unidad en la forma 0 para la unidad A o 1 para la B, y el programa le requerirá los valores deseados para pista y sector. Seguidamente se lee el sector y se muestra la primera mitad del contenido.

Lamentablemente, los 512 bytes de un sector no pueden ser llevados a pantalla como volcado ASCII y Hexadecimal. Por tal motivo puede usted pasearse de una a otra página de la pantalla mediante las teclas 'Shift Cursor Left' y 'Shift Cursor Right'.

El comando 'M' está pensado para modificar la información leída. Este comando pregunta por la dirección de la dirección del buffer a variar. Si responde a esta pregunta simplemente con ENTER, se elige como dirección el comienzo del buffer.

La dirección será mostrada junto con el contenido. El programa espera después el nuevo valor de la dirección de buffer. Las entradas admitidas son los números hexadecimales de 00 a FF, que han de ser entrados sin '&'amp; y terminados mediante ENTER. Después se muestra la dirección de buffer siguiente y su contenido. Pero si no desea modificar dicho contenido, puede saltar a la dirección siguiente pulsando la tecla ENTER. El modo de entrada se abandona con 'X (ENTER)'.

Puede volver a escribir en disco un sector alterado de este modo mediante el comando 'W'. Con la entrada de 'W (ENTER)' se escribe el buffer en el sector del que fue leído en un principio. Pero también puede elegir otro sector si indica, como en el comando 'M', el número de unidad, la pista y el sector.

El comando 'B' supone en cierta medida un medio auxiliar para la lectura de ficheros. Como sabe de capítulos anteriores, en el directorio se almacenan los números de los bloques ocupados por el fichero. La conversión de estos

números de bloque a números de pista y de sector no resulta difícil pero sí engorrosa. El comando 'B' nos libera de esta tarea. Para ello indique el número de bloque hallado en el directorio, el CPC calculará automáticamente los valores correspondientes y los mostrará en la línea de cabecera. Mediante 'R (ENTER)' podrá usted obtener y leer después el primer sector calculado del bloque. Obtendrá el segundo sector pulsando la tecla 'Cursor Right'.

El comando 'C' le muestra un directorio normal del disco insertado. Esta función es bastante práctica ya que no ha de abandonar usted el programa para conseguir una visión general del contenido del disco.

Ya que el programa está escrito en gran parte en BASIC, resulta especialmente indicado para ser ampliado mediante rutinas y comandos propios. Una ampliación posible es la de mostrar el formato del disco introducido y reconocer el cambio de disco. Si leyó un disco en el formato de datos, debería resignarse a enviar antes la orden 'C' en el cambio de formato, para que el AMSDOS determine el formato modificado y modifique correspondientemente el DPB. Podría resolver el problema, por ejemplo desconectando los mensajes de error y comprobando, de la mano de los flags, si el sector fue leído con éxito. Si no es el caso, puede saltar a la rutina del AMSDOS para conseguir el formato (&C56C) y volver a leer el sector.

Otro punto débil que podría usted modificar, es el comando 'B'. Funciona correctamente SOLO en los formatos de CP/M estandar y Vendor. Los resultados son, en otros formatos, erróneos.

Otra posibilidad de ampliación sería la de dar salida por impresora al contenido de sector. Podría servir para ello la rutina de volcado hexadecimal del ejemplo para lectura de un único sector.

El monitor de diskette

1000- *****Discmonitor*****

```
1010 ' *****      RBR 17/4785      *****
1020 '
1030 DEFINT a-l,n-z
1040 MEMORY &A000-1
1050 MODE 2
1060 LOCATE 10,10:PRINT"Un momento por favor .......""
1070 GOSUB 9000: 'definir ventanas y pokear l.m.
1080 CLS #1
1090 befs=" "
1100 buffer = &A2
1110 cmd$="CRWMB"+CHR$(&F0)+CHR$(&F1)+CHR$(&F2)+CHR$(&F3)
1120 cmd$=cmd$+CHR$(&F6)+CHR$(&F7)
1130 FOR i=1 TO LEN(cmd$):cmd1$=cmd1$+MID$(cmd$,i,1)+",":NEXT
1140 cmd1$=LEFT$(cmd1$,LEN(cmd1$)-1)
1150 sector = 1:track = 0:drive = 0:befehl = &84:catflag = 0
1160 '
1170 '
2000 ' ***** programa principal *****
2010 '
2020 POKE &A108,buffer:POKE &A10A,buffer
2030 LOCATE #1,1,1:CLS
2040 GOSUB 6000 : 'mostrar una pagina
2050 PRINT"comando ""cmd1$"" >"
2060 befs=UPPER$(INKEY$):IF befs="" THEN 2060 ELSE CLS
2070 ON INSTR(cmd$,bef$)GOTO 3100,3150,3200,3250,3300,3500,3
550,3600,3650,3400,3450
2080 GOTO 2050
2090 '
2100 '
3000 ' ***** tabla de comandos *****
3010 '
3100 ' ***** comando-d *****
3110 GOSUB 4000 : GOTO 2050
3120 '
3150 ' ***** comando-r *****
3160 GOSUB 5000 : GOTO 2020
```

```
3170,
3200 ' ***** comando-w *****
3210 GOSUB 5100 : GOTO 2020
3220 '
3250 ' ***** comando-m *****
3260 GOSUB 7000 : GOTO 2030
3270 '
3300 ' ***** comando-b *****
3310 GOSUB 8000 : GOTO 2030
3320 '
3400 ' ***** mostrar primera pagina *****
3410 POKE &A108,buffer:POKE &A10A,buffer:GOTO 2030
3420 '
3450 ' ***** mostrar segunda pagina *****
3460 POKE &A108,buffer + 1:POKE &A10A,buffer + 1:GOTO 2030
3470 '
3500 ' ***** track + 1 *****
3510 track=-(track<39)+track+((track=39)*39):GOSUB 5500:GOTO
2020
3520 '
3550 ' ***** track - 1 *****
3560 track=(track>0)+track+(-(track=0)*39):GOSUB 5500:GOTO 2
020
3570 '
3600 ' ***** sector - 1 *****
3610 sector=(sector>1)+sector+(-(sector=1)*(PEEK(&ABA0+(drive*64))-1))
3620 IF 'sector=9 THEN GOTO 3550 ELSE GOSUB 5500:GOTO 2020
3630 '
3650 ' ***** sector + 1 *****
3660 sector=-(sector<9)+sector+((sector=9)*(PEEK(&ABA0+(drive*64))-1))
3670 IF sector=1 THEN GOTO 3500 ELSE GOSUB 5500:GOTO 2020.
3680 '
4000 ' ***** mostrar directorio *****
4010 WINDOW SWAP 0,1:CLS:PRINT:DIR:WINDOW SWAP 1,0:catflag
= 1
4020 RETURN
4030 '
5000 ' ***** leer sector *****
5010 PRINT"      leer sector":PRINT
5020 GOSUB 5800:GOTO 5500
5030 '
5100 ' ***** escribir sector *****
5110 befehl = &85:PRINT"      escribir sector":PRINT
```

```
5120 GOSUB 5800
5130 '
5500 ' ***** leer/escribir sector *****
5510 POKE &A100,befehl
5520 POKE &A104,drive
5530 POKE &A105,track
5540 POKE &A106,sector-1 + PEEK(&A89F+drive*&40)
5550 CALL &A0A0:befehl = &84:RETURN
5560 '
5800 ' ***** elegir disco, pista y sector *****
5810 INPUT"      disco (0/1) o bien enter";drive$:IF drive$=
"" THEN RETURN
5820 drive = VAL(drive$)
5830 INPUT"      pista (0-39)           ";track
5840 INPUT"      sector (1-9)          ";sector
5850 RETURN
5860 '
6000 ' ***** mostrar *****
6010 IF catflag = 1 THEN catflag = 0:CLS #1
6020 WINDOW SWAP 0,1
6030 PRINT USING"      ; disco ###      ; pista ###      ; se
ctor ###      ";drive,track,sector
6040 PRINT
6050 IF INKEY$ <> "" GOTO 6070
6060 IF INSTR("BW",bef$) = 0 THEN CALL &BB81:CALL &A000:CALL
&BB84
6070 WINDOW SWAP 1,0
6080 RETURN
6090 '
7000 ' ***** modificar *****
7010 PRINT"      modificar buffer"
7020 INPUT"direccion de buffer ";buadr$
7030 IF buadr$ = "" then buadr=0 ELSE buadr=VAL("&"&buadr$)
7040 madr=buadr+(buffer*256)
7050 PRINT HEX$(buadr,4);";HEX$(PEEK(madr),2);";"
7060 INPUT newbyt$:newbyt$=UPPER$(newbyt$)
7070 IF newbyt$="X" THEN RETURN
7080 IF newbyt$="" THEN GOTO 7110
7090 newbyt=VAL("&"&newbyt$):newbyt$=""
7100 POKE madr,newbyt
```

```
7110 buadr=buadr+1:GOTO 7040
7120 '
8000 ' ***** convertir numero bloque *****
8010 PRINT"      numero de bloque (amsdos-format)"
8020 INPUT"      numero de bloque (en hex)    ";block$
8030 block$=""&block$:a=VAL(block$)
8040 sectcnt=a*2+18
8050 track = INT((a*2+18)/9)
8060 sector = (a*2+18) MOD 9 + 1
8070 RETURN
8080 '
9000 ' ***** definir ventanas *****
9010 WINDOW #0,1,80,20,25
9020 WINDOW #1,1,80,1,19
9030 '
9100 ' ***** DATAS PARA HEXDUMP/SECTOR I-O *****
9110 FOR i = &A000 TO &AOBC
9120 READ byte : POKe i,byte : s = s + byte : NEXT
9130 DATA &C3,&2D,&A0,&7C,&CD,&0B,&A0,&7D,&F5,&1F,&1F,&1F,&1
F,&CD,&11,&A0
9140 DATA &F1,&E6,&0F,&C6,&30,&FE,&3A,&3B,&02,&C6,&07,&C3,&5
A,&BB,&3E,&0D
9150 DATA &CD,&5A,&BB,&3E,&0A,&C3,&5A,&BB,&3E,&20,&C3,&5A,&B
B,&2A,&07,&A1
9160 DATA &ED,&5B,&09,&A1,&FD,&21,&01,&00,&06,&10,&E5,&C5,&E
D,&4B,&0B,&A1
9170 DATA &09,&CD,&03,&A0,&C1,&E1,&CD,&28,&A0,&CD,&2B,&A0,&7
E,&CD,&0B,&A0
9180 DATA &FD,&2B,&CD,&2B,&A0,&A7,&ED,&52,&19,&2B,&2C,&23,&1
0,&EE,&01,&FO
9190 DATA &FF,&09,&06,&10,&CD,&2B,&A0,&7E,&E6,&7F,&FE,&20,&3
8,&02,&1B,&02
9200 DATA &3E,&2E,&CD,&5A,&BB,&23,&10,&EF,&CD,&1E,&A0,&E5,&3
7,&ED,&52,&E1
9210 DATA &CB,&FD,&E5,&C1,&0D,&1B,&B1,&05,&3E,&10,&90,&4F,&0
6,&00,&ED,&42
9220 DATA &23,&41,&1B,&D0,&4E,&4B,&3A,&4C,&44,&09,&41,&2C,&2
2,&20,&22,&0D
9230 DATA &21,&00,&A1,&CD,&D4,&BC,&22,&01,&A1,&79,&32,&03,&A
1,&21,&04,&A1
```

```
9240 DATA &5E,&23,&56,&23,&4E,&21,&00,&A2,&DF,&01,&A1,&C9,&2
D
9250 IF s <> 20027 THEN PRINT "error en suma de prueba" : E
ND
9260 S=0
9270 FOR i = &A100 TO &A10C
9280 READ byte : POKЕ i,byte : s = s + byte : NEXT
9290 DATA &B4,&00,&00,&00,&00,&00,&01,&00,&A2,&FF,&A2,&00,&5
E
9300 IF s <> 806 THEN PRINT "error en suma de prueba" : END
9310 RETURN
```

5.6 El manager de discos

El manager de disco sirve para la creación y cuidado de discos de trabajo. Ofrece las diversas funciones en los puntos de un menú de modo que, por ejemplo para formatear un disco ya no ha de cargar el CP/M. Tras teclear el manager de disco, debería asegurarlo almacenándolo en un diskette. Tras iniciar el programa, obtendrá el siguiente menú en pantalla:

- 1) Modificar nombre de ficheros
- 2) Borrar ficheros
- 3) Formatear un disco
- 4) Mostrar directorio
- 5) Copiar FICHEROS DE NO-PROGRAMA
- 6) APPEND - Encadenar 2 ficheros
- 7) Mostrar contenido de un fichero (TYPE)

- 9) Fin del programa

Para elegir un punto del menú pulse simplemente en el teclado la cifra correspondiente (sin tecla ENTER). Inmediatamente entrará en la rutina respectiva.

Para simplificar la explicación, trataremos uno a uno los puntos del menú:

1) Modificar nombre de ficheros

Este punto del menú le ayudará a modificar el nombre de un fichero. Se le pregunta el nombre del fichero que desea modificar. Introduzca el nombre correspondiente y pulse la tecla ENTER.

Si desea abandonar un punto del menú, pulse la tecla ENTER sin dato alguno en la entrada de cualquier campo.

Tras haber dado el nombre del fichero, se le preguntará por el nuevo nombre. Si, por ejemplo, desea cambiar el nombre de un fichero "A" por el nombre "B", introduzca "A" como nombre viejo y "B" como nombre nuevo.

Se le pedirá que introduzca el disco en el que se halla el fichero que usted desea renombrar. Tras pulsar una tecla, el nombre del fichero es modificado en el disco.

2) Borrar ficheros

Este punto es el más cómodo del manager de discos. Sirve para "ordenar" discos. Quedará asombrado de todo lo que llega a reunirse en un disco, entre programas y datos. Con esta rutina puede liberar a su disco de programas y datos viejos y nada importantes, de una forma cómoda.

Tras mostrar el directorio en pantalla, puede "recorrer" con el cursor los distintos nombres de fichero. Pulsando la tecla COPY puede marcar un fichero para borrado - el campo correspondiente intermitirá de rojo a rosa - o retirar una tal marca.

Cuando haya marcado todos los ficheros que deseé borrar, pulse la tecla ENTER. Se le volverá a preguntar si ha marcado todos los ficheros que desea borrar y si desea realmente borrarlos. Tras confirmar estas preguntas, los ficheros marcados son borrados del disco.

A nosotros en particular, esta rutina se nos ha mostrado muy útil para depurar discos, sobre todo por no ser tan "lanzada" como el comando IERA, ða\$ con el se borran muy fácilmente ficheros que no pretendíamos eliminar.

3) Formatear un disco

Este punto del menú es especialmente interesante puesto que no hay otro modo de formatear un disco desde el BASIC, y todos los programas que sean confortables deberían ofrecer este punto del menú.

Esta rutina de formatear es más rápida que la rutina bajo CP/M; sin embargo se renuncia a una consulta de seguridad, es decir, no se comprueba cada sector formateado. Nosotros no hemos tenido nunca problemas con discos formateados mediante esta rutina. No obstante, si alguien desea tener completa seguridad, debería emplear la rutina CP/M FORMAT.

Puede formatear en formato Vendor y en formato de sólo datos; no está previsto el formateo en formato IBM.

El formato Vendor significa que el disco es formateado como disco de sistema CP/M en el que sin embargo no se halla el CP/M, pero que podrían copiarse posteriormente.

Lo normal sería formatear su disco en el formato Data-Only-Format; podría así aprovechar la capacidad total del disco para el almacenamiento de datos y programas BASIC. Tras elegir el formato, se le pide que introduzca el disco a formatear y que pulse una tecla cualquiera.

5) Copiar FICHEROS DE NO-PROGRAMA

Mediante esta rutina no se pueden copiar ficheros de programa, pero si ficheros de datos generados por programas. Puede elegir entre copiar los ficheros fuente y objeto en el mismo disco o en discos distintos. Si los ficheros fuente y objeto se hallan en el mismo disco, no existen limitaciones respecto a la longitud de los ficheros en si. Si se copia de un disco a otro, sólo es posible un máximo de 250 juegos de datos. En caso de que el fichero a copiar sea demasiado largo, ello le será comunicado en pantalla.

6) APPEND - Encadenar 2 ficheros

Este punto del menú puede convertir dos ficheros en uno. Se le pregunta por los nombres de los dos ficheros que han de ser encadenados: el primer fichero supondrá la primera parte del nuevo fichero, el segundo fichero irá encadenado detrás. El fichero objeto será creado en el mismo disco en el que se hallan ambos ficheros fuente.

7) Mostrar contenido de un fichero

Este punto del menú hace lo mismo que el comando TYPE bajo CP/M - le muestra en pantalla el contenido de un fichero ASCII. Pulsando cualquier tecla puede detener y reemprender el proceso.

Estoy convencido de que estas rutinas le proporcionarán un buen servicio en su tarea cotidiana con la unidad de disco Amstrad.

```
10 '*****  
20 *** Manager de discos ---- JS/RB 1.5.85 ***  
30 *****  
40 :  
50 OPENOUT "dummy" : MEMORY HIMEM-1 : CLOSEOUT  
60 DATA &3e,&00,&32,&2f,&80,&3a,&2f,&80,&57,&3a,&30,&80,&5f,  
&3a,&31,&80  
70 DATA &4f,&21,&35,&80,&df,&32,&80,&3a,&2f,&80,&fe,&27,&c8,  
&3c,&32,&2f  
80 DATA &80,&21,&35,&80,&06,&09,&77,&23,&23,&23,&23,&10,&f9,  
&18,&d6,&27  
90 DATA &00,&41,&52,&c6,&07  
100 :  
110 FOR i=&8000 TO &8034  
120 READ d  
130 POKE i,d  
140 s=s+d  
150 NEXT  
160 IF s<>4258 THEN PRINT"*** Error en Datas ***":END  
170 MEMORY &7FFF  
180 :  
190 CLEAR : DEFINT b-z  
200 MODE 1 : INK 0,11 : INK 1,16,6 : INK 2,0 : INK 3,24 : PE  
N 3 : PAPER 2 : CLS  
210 BORDER 0  
220 CLS : ORIGIN 0,0,0,640,340,400 : CLG 3  
230 PAPER 3 : PEN 2 : LOCATE 14,2 : PRINT"Manager de discos"  
: PAPER 2 : PEN 3.  
240 LOCATE 1,7  
250 PRINT"1) Modificar nombre de ficheros"  
260 PRINT"2) Borrar ficheros"  
270 PRINT"3) Formatear un disco"  
280 PRINT"4) Mostrar directorio"  
290 PRINT"5) Copiar FICHEROS DE UN PROGRAMA  
300 PRINT"6) APPEND - Encadenar 2 ficheros"  
310 PRINT"7) Mostrar contenido de un fichero"  
320 PRINT : PRINT"9) Fin de programa"  
330 LOCATE 1,20 : PEN 1 : PRINT "Su elección:"
```

```
340 a$=INKEY$ : IF a$="" THEN 340
350 we=VAL(a$) : IF we<1 OR (we>7 AND we<>9) THEN 340
360 PEN 3 : ON we GOTO 1200,550,380,1120,1340,1700,1900,1900
,2100
370 '
380 '=====
390 '      Formatear disco
400 '=====
410 :
420 CLS : PRINT"1) Formato Vendor" : PRINT : PRINT"o" : PRIN
T : PRINT"2) Formato Data-Only"
430 a$=INKEY$ : IF a$<"1" OR a$>"2" THEN 430
440 IF a$="1" THEN f$="Vendor" : y=&41 ELSE f$="Data-Only" :
y=&C1
450 x=&8035
460 FOR i=1 TO 9
470   POKE x,0 : POKE x+1,0 : POKE x+2,y : POKE x+3,2
480   x=x+4
490   y=y+2 : IF (y AND &F) = &B THEN y=y-9
500 NEXT
510 PRINT : PRINT"Por favor, introduzca el disco" : PRINT"y
pulse una tecla..."
520 IF INKEY$="" THEN 520
530 CALL &8000 : GOTO 190
540 '
550 '=====
560 '      Borrar Ficheros
570 '=====
580 :
590 DIM a$(65),era(64) :CLS
600 LOCATE 4,11 : PRINT"Se esta leyendo el directorio ..."
610 PRINT : PRINT"      un momento, por favor"
620 lin$=STRING$(40,154)
630 FOR i=0 TO 63
640   a$(i)=STRING$(11,32)
650 NEXT
660 a=PEEK(&BB5A) : POKE &BB5A,&C9 : CAT : POKE &BB5A,a
670 anz=PEEK(&A912) : a=PEEK(&A79C)*256 + PEEK(&A79B)+1
680 CLS
690 FOR i=0 TO anz
```

```
700   POKE @a$(i)+1,a-(INT(a/256)*256)
710   POKE @a$(i)+2,INT(a/256) : a=a+14
720 NEXT
730 FOR i=0 TO anz
740 IF ASC(LEFT$(a$(i),1)) = 0 THEN a=i : i=anz : GOTO 770
750 a$(i)=LEFT$(a$(i),8) + "." + RIGHT$(a$(i),3)
760 PRINT a$(i),
770 NEXT : anz=a
780 LOCATE 1,22 : PRINT lin$
790 txt$="Borrar en este disco? (S/N)" : GOSUB 1090
800 GOSUB 1100 : IF LOWER$(a$)="s" THEN 840
810 txt$="ENTER = nuevo disco, X = Fin" : GOSUB 1090
820 GOSUB 1100 : IF a$=CHR$(13) THEN ERASE a$,era : GOTO 550
830 IF LOWER$(a$)="x" THEN 190 ELSE 820
840 txt$="Teclea 'COPY' marca para borrar" : GOSUB 1090
850 x=0 : xc=1 : yc=1 : GOTO 950
860 x=temp : GOSUB 1100 : IF a$=CHR$(13) THEN 1000
870 IF ASC(a$)=&FO THEN x=x-3 : IF x<0 THEN 860 ELSE 950
880 IF ASC(a$)=&F1 THEN x=x+3 : IF x>anz-1 THEN 860 ELSE 950
890 IF ASC(a$)=&F2 THEN x=x-1 : IF x<0 THEN 860 ELSE 950
900 IF ASC(a$)=&F3 THEN x=x+1 : IF x>anz-1 THEN 860 ELSE 950
910 IF ASC(a$)<>&EO THEN 860
920 era(x) = era(x) XOR 1 'Invierte campo
930 LOCATE xc,yc : PAPER era(x)
940 PRINT a$(x); : PAPER 2 : GOTO 860
950 yco = x\3+1 : xco = (x- ((yco-1)*3))*13+1
960 LOCATE xc,yc : PAPER (era(temp)=0)*-2+era(temp)
970 PRINT a$(temp); : PAPER 2
980 LOCATE xco,yco : PAPER era(x) : PRINT a$(x); : PAPER 2
990 xc=xco : yc=yco : temp=x : GOTO 860
1000 LOCATE xc,yc : PAPER 2+(era(x)<>0) : PRINT a$(x); : PAPER 2
txt$="Todos los ficheros marcados? (S/N)" : GOSUB 1090
1010 GOSUB 1100 : IF LOWER$(a$)<>"s" THEN 840
1020 txt$="Borrar efectivamente (S/N)" : GOSUB 1090
1030 GOSUB 1100 : IF LOWER$(a$)<>"s" THEN 840
1040 txt$="Se borran los ficheros !" : GOSUB 1090
1050 FOR i=0 TO anz
1060 IF era(i) THEN !ERA,@a$(i)
1070 NEXT
1080 GOTO 190
```

```
1090 LOCATE 4,24 : PRINT CHR$(20);txt$ : RETURN
1100 a$=INKEY$ : IF a$="" THEN 1100 ELSE RETURN
1110 '
1120 =====
1130 ' Muestra contenido disco
1140 =====
1150 :
1160 CLS : CAT
1170 PRINT : PRINT"Pulsacion de tecla ..."
1180 IF INKEY$="" THEN 1180 ELSE 190
1190 :
1200 =====
1210 ' Modificar nombre de fichero
1220 =====
1230 :
1240 CLS
1250 INPUT "Nombre del antiguo fichero : ",falt$ : IF falt$=
"" THEN 190
1260 INPUT "Nuevo nombre del fichero : ",Neu$ : IF neu$="" T
HEN 190
1270 PRINT : PRINT"For favor, introduzca el disco"
1280 PRINT : PRINT"en el que se halla el fichero,"CHR$(24);f
alt$;CHR$(24)
1290 PRINT : PRINT"y pulse una tecla"
1300 IF INKEY$="" THEN 1300
1310 !REN,@neu$,@falt$
1320 GOTO 190
1330 '
1340 =====
1350 ' Copia fichero de no-programa
1360 =====
1370 :
1380 CLS : INPUT "Nombre del fichero fuente : ",quell$ : IF
quell$="" THEN 190
1390 INPUT "Nombre del fichero de destino : ",ziel$ : IF zie
l$="" THEN 190
1400 INPUT "Copiar en otro disco (S/N) : ",jn$
1410 IF LOWER$(jn$)="n" THEN 1620
1420 DEFSTR a : DIM a(250)
1430 PRINT : PRINT"For favor, introduzca el disco fuente"
1440 PRINT"y pulse una tecla"
```

```
1450 IF INKEY$="" THEN 1450
1460 OPENIN quell$
1470 WHILE NOT EOF
1480   LINE INPUT #9,a(xc)
1490   xc=xc+1 : IF xc>250 OR FRE(0)<2000 THEN PRINT CHR$(7)
"El fichero es demasiado grande" : FOR i=1 TO 1000:NEXT:CLOS
EIN:RUN 190
1500 WEND
1510 CLOSEIN
1520 PRINT : PRINT CHR$(7)"Por favor, introduzca el disco de
stino"
1530 PRINT"y pulse una tecla"
1540 IF INKEY$="" THEN 1540
1550 PRINT : PRINT"Se copian "xc" juegos de datos."
1560 OPENOUT ziel$
1570 FOR i=0 TO xc
1580   PRINT #9,a(i); : IF LEN(a(i))<255 THEN PRINT #9
1590 NEXT
1600 CLOSEOUT
1610 RUN 190
1620 OPENIN quell$ : OPENOUT ziel$
1630 WHILE NOT EOF
1640   LINE INPUT #9,a$
1650   PRINT #9,a$
1660 WEND
1670 CLOSEIN : CLOSEOUT
1680 GOTO 190
1690 '
1700 =====
1710 ' Encadena dos ficheros
1720 =====
1730 '
1740 CLS: INPUT "Nombre del primer fichero : ",f$(0) : IF f
$(0)="" THEN 190
1750 INPUT "Nombre del segundo fichero : ",f$(1) : IF f$(1)=
"" THEN 190
1760 INPUT "Nombre del fichero destino : ",f3$ : IF f3$="" T
HEN 190
1770 PRINT : PRINT"ok"
1780 OPENOUT f3$
1790 FOR i=0 TO 1
```

```
1800  OPENIN f$(i)
1810  WHILE NOT EOF
1820    LINE INPUT #9,a$
1830    PRINT #9,a$
1840  WEND
1850  CLOSEIN
1860 NEXT
1870 CLOSEOUT
1880 GOTO 190
1890 '
1900 =====
1910 ' Muestra contenido de un fichero
1920 =====
1930 :
1940 ON ERROR GOTO 2090
1950 CLS : INPUT "Nombre del fichero : ",f$ : IF f$="" THEN
190
1960 MODE 2 : PEN 1 : INK 1,1 : PRINT"Contenido del fichero
" f$
1970 PRINT STRING$(80,"-")
1980 OPENIN f$
1990 WHILE NOT EOF
2000  LINE INPUT #9,a$
2010  PRINT a$
2020  IF INKEY$="" THEN 2040
2030  IF INKEY$="" THEN 2030
2040 WEND
2050 CLOSEIN
2060 PRINT : PRINT"Pulsacion de tecla"
2070 IF INKEY$="" THEN 2070
2080 GOTO 190
2090 PRINT"File Type Error - El fichero no es ASCII !!!" : RE
SUME 2060
2100 CLS : PEN 1 : PRINT"Programa finalizado
```

Indice alfabético**A**

A	63
A:	34
ACCESO DIRECTO	75, 108, 109, 350
ALMACENAMIENTO	61
ALMACENAMIENTO RELATIVO	37, 108, 350
AMSDOS	26, 32, 34, 46, 62, 91, 96
AMSDOS	125, 139, 151, 167, 235, 247, 329
AMPLIACION DE ORDENES	145
AMPLIACIONES	356
APPEND	108
ARCHIVO DE TARJETAS	109
ASSEMBLER	146

B

B	63
B:	34
BAS	59
BASIC	21
BAUD	14
BLOQUE	61
BOOTEN	34, 45
BOOTGEN	34, 45
BORRAR	40
BUFFER	93, 96
BUFFER DE ENTRADA	96
BUFFER DE SALIDA	96

C

CABEZAL DE ESCRITURA	17
CABEZAL DE LECTURA	17
CANAL	96
CANCEL	27
CAPACIDAD DE MEMORIA	18
CARRIAGE RETURN	77, 98
CAS CATALOG	128
CAS IN ABANDON	127
CAS IN CLOSE	127
CAS IN DIRECT	127
CAS IN OPEN	127
CAS OUT ABANDON	127
CAS OUT CHAR	127
CAS OUT CLOSE	127
CAS OUT DIRECT	128
CAS OUT OPEN	127
CAS RETURN	127
CAS TEST EOF	127
CASSETTE	13
CASSETTE, UNIDAD DE	14, 74
CAT	49, 57, 125
CHAIN MERGE	49, 49, 329
CHKDISC	30
CHKDISK	34
CLOAD	34
CLOSE	140
CLOSEIN	49
CLOSEOUT	49
COMANDOS	23
COMANDOS AMSDOS	62
COMANDOS CP/M	23
COMANDOS OCULTOS	151
COMODIN	38
COMPATIBILIDAD	123
COMPROBACION	30

CON	42
CONMUTAR UNIDAD DE DISCO	41
COPIA	27, 35
COPYDISC	29, 34
CP/M 2.2	22
CP/M 3.0	17, 18, 21, 22
CPC	119
CSAVE	34

D

DATA-ONLY-FORMAT	33
DATOS, ALMACENAMIENTO DE	37, 74, 108
DATOS, ALMACENAMIENTO RELATIVO DE	350
DATOS, CAMPO DE	76, 77, 350, 378
DATOS, FORMATO DE	211
DATOS, JUEGO DE	75, 76, 77, 109, 350, 378
DBASE	21
DERR	123
DESARROLLO	16
DIAGRAMA DE SINTAXIS	88
DIFERENCIAS	119
DISPOSICION DE CONEXIONES	172
DIR	34, 39, 57, 63
DIRECCION DE VARIABLE	71
DIRECTORIO	39, 212
DIRECTORY	23, 39
DISC	63
DISC CATALOG	212
DISC FORMAT PARAMETER	142
DISC IN ABANDON	155
DISC IN CHAR	106, 132
DISC IN CLOSE	131
DISC OUT ABANDON	140
DISC OUT CHAR	140, 141
DISC OUT CLOSE	139
DISC OUT OPEN	136

DISC RETURN	134
DISC TEST EOF	49, 63, 134
DISC-CONTROLLER	125
DISC.IN	49, 63
DISC.OUT	49, 64
DISCCCHK	30, 34
DISCCOPY	28, 34
DISCO, UNIDAD DE	13, 16
DISCO, MONITOR DE	395
DISCOS DE TRES PULGADAS	16
DISC IN OPEN	128
DISKETTE o DISCO	16, 24
DISKETTE, INCORPORADO	20
DISKETTE, ERRORES DE	119, 122
DISKETTE, FORMATO DE	209
DISKETTE, FORMATOS DE	198
DR. LOGO	20
DRIVE PARAMETER	49, 64, 51

E

ED	34
END OF FILE	132
EOF	49, 83, 85, 132
ERA	34, 40, 49, 64
ERR	123
ERRORES	329
ERROR, MENSAJES DE	329, 336
ESCRITURA	160

F

FACTOR DE DISTORSION	220
FDC	164, 167
FDC 765	170
FDC - PROGRAMACION	180

FICHERO	15, 74, 75, 87, 109, 378
FICHERO ASCII	139
FICHERO BINARIO	96
FICHERO DE BACKUP	59
FICHERO DE PROGRAMA	74
FICHERO SECUENCIAL	79, 87
FICHERO, INDICATIVO DE	36
FICHERO, NOMBRE DE	51
FICHERO, ORDENES DE	54
FICHERO, PROGRAMA DE GESTION DE	378
FICHERO, TIPO DE	36, 219
FICHEROS KEY-INDEX	119
FILECOPY	28, 34, 35, 38
FIRMWARE-MANUAL	127
FLOPPY	13
FLOPPY, BUFFER DEL	93, 96
FLOPPY, CONTROLADOR DEL	167
FLOPPY, INTERFACE DEL	167
FORMAT	30, 34, 161
FORMATEADO	18, 24
FORMATO IBM	33, 211
FORMATOS	32
FORMATOS DE FORMATEADO	31

G

GAP	228
GARBAGE COLLECTION	93
GET	106

H

HEAD LOAD TIME	152
HEAD LOADE TIME	154
HEADER	219
HIMEM	93

I

ID	228
IGNOR	27
INDICATIVO	219
INPUT	49, 103
INSTRUCCION DIM	86
INTERFACE	169
ISAM	119

J

JUEGO DE DATOS	75
----------------------	----

K

KERNAL	152
KL FAR PCHL	152
KL FIND COMMAND	146

L

LINE INPUT	49, 103
LIST	49
LOAD	8, 49
LOGO	17, 23
LST	42

M

MAGNETISMO	220
MANUAL	15
MARCA DE DIRECCION	228
MEMORIA EXTERNA	16
MEMORIA TEMPORAL	13
MEMORY	93
MERGE	49, 329
MESSAGE ON/OFF	151
MFM	220
MODOS DE FUNCIONAMIENTO	117
MOVCPM	34, 45

N

NUMERO DE RECORD	115
------------------------	-----

O

ON-ERROR-GOTO	122, 151, 336
OPENIN	49
OPENOUT	49, 125
ORDEN DE LECTURA	82
ORDENADORES CPC	13
ORDENES EXTERNAS	62

P

PASCAL	87, 115
PATCHEAR	143
PERFORACION DE INDICE	16, 188

PIP	34, 42
PISTA	24, 25, 163
PREPARACION DE UN FICHERO	110
PRINT	82, 83, 108
PROCESADOR	21
PROGRAMA	49, 98
PROGRAMA MAQUINA	61, 125
PROGRAMACION	79
POINTER INTERNO	108
PUN	42
PUNTERO	82, 83, 108
PUNTERO INTERNO	82, 108

R

RAM	119, 125, 210, 235
RANDOM ACCESS	108, 109
RDR	42
READ SECTOR	156
READ-ONLY	43, 218
READ-WRITE	43
RECALIBRADO	193
RECORD	26, 115, 356
REGISTRO DE ESTADO	198
RELATIVO	37, 108
REN	34, 41, 49, 68
RENOMBRAR	41
REPRESENTACION	41
RESIDENTE	34
RESTART	152
RETORNO DE CARRO	77, 98
RETRY COUNT	27, 164
ROM	62, 119, 125, 210, 235, 247
ROM DEL AMSDOS	247
ROM, LISTADO DE LA	249
RSX	146, 356, 378

S

SAVE	34, 49, 59
SECTOR	24, 25, 157, 159, 160
SECUENCIA	74
SECUENCIAL	37, 74
SECUENCIAS DE CONTROL	31
SEEK TRACK	163, 193
SETUP	46
SIMBOLO SEPARADOR	77, 85, 98
SINTAXIS	87
SISTEMA OPERATIVO	53, 54, 69
SISTEMA, DESPLAZAMIENTO DEL	45
SISTEMA, DISCO DE	247
SISTEMA, DISCO DEL	22
SISTEMA, FORMATO DE	210
SISTEMA, RAM DE	235
SOBREESCRITURA	17
SOFTWARE	21
SPEED WRITE	14, 125
STAT	34, 43
STRINGS	93
SYSGEN	34, 45

T

TAPE	49, 69
TAPE.IN	49, 69
TAPE.OUT	49, 69
TECNICA	167
TERMINAL COUNT	181
TEST DRIVE	164
TRACK	163
TRANSEUNTE	34
TTL	167
TYPE	34, 41

U

UNIDAD DE DISCO	51
USER	34, 49, 53, 69
USER, ZONA DE	53

V

VARIABLE DE CAMPO	85, 87, 88
VECTOR	143
VECTORES DOS	125
VERSIONES CPM	22

W

WILD CARDS	38
WIRTH	87
WORDSTAR	21
WRITE SECTOR	49, 98, 160

Z

Z-80	21
------------	----

EL CONTENIDO:

EL GRAN LIBRO DEL FLOPPY CPC lo explica todo sobre la programación con discos y la gestión relativa de ficheros mediante el floppy DDI-1 y la unidad de discos incorporada del CPC 664/6128. La presente obra, un auténtico estándar, representa una ayuda incomparable tanto para el que desee iniciarse en la programación con discos como para el más curtido programador de ensamblador. Especialmente interesante resulta el listado exhaustivamente comentado del DOS y los muchos programas de ejemplo, entre los que se incluye un completo paquete de gestión de ficheros.

Extracto del contenido:

- Todo sobre ficheros secuenciales
- Listado de una gestión de ficheros
- Intercepción de mensajes de error
- Gestión relativa de ficheros (!)
- Listado DOS comentado
- Disk-monitor
- Gestor de discos
- Siete nuevos comandos de disco
- Acceso directo
- Fundamentos de CP/M
- y mucho más.

ESTE LIBRO HA SIDO ESCRITO POR:

Rolf Brückmann, entusiasta técnico y programador de sistemas en la casa DATA BECKER, conocido por una multitud de obras estándar (64 interno, CPC interno...). El es el responsable de la parte técnica de la presente obra. Jörg Schieb es un experto en programación de ficheros y posee un dominio absoluto de la programación en lenguaje de máquina.

ISBN 84-86437-58-X