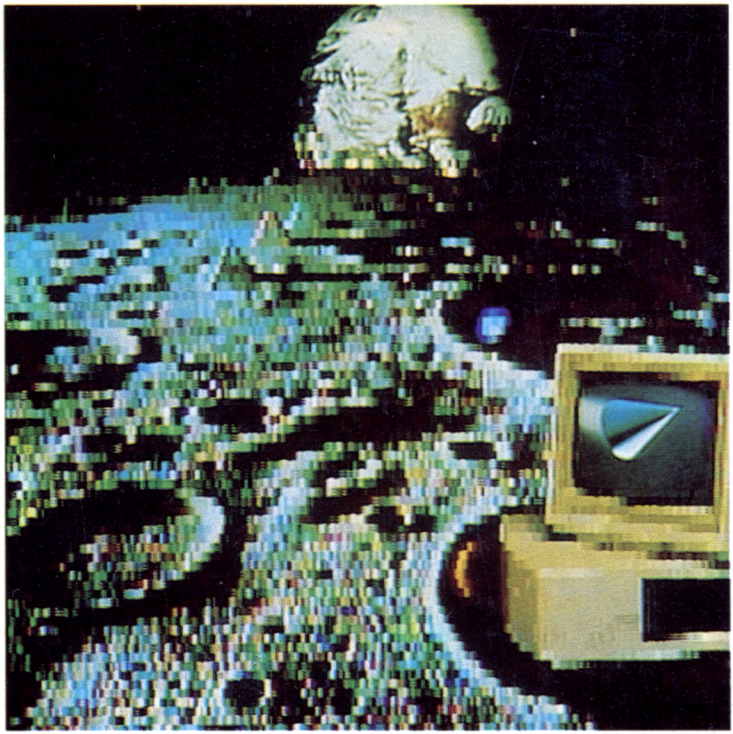
GRAN BIBLIOTECA

AMSTRAD



LA UNIDAD DE DISKETTE

UNA ENCICLOPEDIA EN 3 PULGADAS

EDICIONES INGELEK

GRAN BIBLIOTECA AMSTRAD

20

LA UNIDAD DE DISKETTE

Director editor:
Antonio M.ª Ferrer Abelló

Director de producción:
Vicente Robles

Director de la obra:
Fernando López Martínez

Redactor técnico:
Antonio Manzanera Amaro
Yolanda Toledano Sobremazas

Colaboradores:
L-H Servicios Informáticos
Pilar Manzanera Amaro
Carlos de la Ossa Villacañas

Diseño:
Bravo/Lofish

Maquetación:
Carlos González Amezúa

Dibujos:
José Ochoa

Fotografía:
Grupo Gálata

ISBN del tomo: 84-7708-115-8.
ISBN de la obra: 847708-004-6.
Fotocomposición: Andueza, S. A.
Imprime: Eurosur, S. A.
Depósito legal: M-25303-1987
Precio en Canarias, Ceuta y Melilla: 435 ptas.

Octubre 1987

LA UNIDAD DE DISKETTE

INTRODUCCIÓN

os sistemas operativos de los ordenadores CPC, en principio no están dispuestos para soportar la unidad de disco. Solamente a través de una ROM externa son capaces de gestionar este medio de almacenamiento.

Sin embargo, cuando la unidad de disco está conectada a un CPC, la situación cambia radicalmente: órdenes como **LOAD, SAVE** o **CAT,** se dirigen automáticamente hacia este periférico, en lugar de al casete.

En la ROM de AMSDOS están escritas las nuevas rutinas que gestionan estos comandos, no sólo los mencionados anteriormente, sino también las extensiones del sistema residente (RSX) como **|DIR, |ERA,** o **|RENAME,** sin sentido en los sistemas de cinta.

La ROM de AMSDOS llega todavía más lejos: casi por sorpresa, en esta zona de memoria se encuentran también las instrucciones que gestionan nueve órdenes adicionales, a las cuales se puede acceder solamente desde el código máquina, estando oculta su entrada al programador en BASIC.

Para activarlas se siguen los mismos principios (en lenguaje máquina, por supuesto), que con las RSX de disco, y están encargadas de llevar a cabo tareas tan interesantes como la lectura o escritura de un sector del disco, el formateado de una pista, o el posicionamiento del cabezal sobre una pista concreta.

Este volumen de la GRAN BIBLIOTECA AMSTRAD describirá con todo lujo de detalles las interioridades del periférico de almacenamiento masivo por excelencia: la unidad de disco.

Comenzaremos ocupándonos de las funciones de carga, grabación, etc., conocidas por todos cuando las manejamos desde BASIC; a continuación, aprenderemos a controlar las extensiones del sistema de disco (RSX), analizando a fondo las nuevas órdenes, que mejor podríamos denominar «funciones ocultas» de AMSDOS. Para finalizar, una amplia descripción del chip controlador de disco, nos abrirá las puertas de un mundo de posibilidades ante esa auténtica enciclopedia en tres pulgadas que es un diskette.

COMANDOS DEL AMSDOS

l AMSDOS es el sistema operativo de disco en los ordenadores CPC. Mediante él, podemos controlar las operaciones del diskette a partir del BASIC.

Existen dos conjuntos de órdenes para el disco.

El primer conjunto de órdenes es válido tanto para el disco como para el casete y se conoce bajo la denominación de ORDENES RESIDENTES, el cual está compuesto por:

CAT
CHAIN
CHAIN MERGE
CLOSEIN
CLOSEOUT
EOF
INPUT#9
LINE INPUT#9
LIST#9
LOAD
MERGE
OPENIN
OPENOUT
PRINT#9
RUN
SAVE
WRITE#9

El segundo conjunto de órdenes, conocidas como ORDENES EXTERNAS, está constituido por diez comandos que residen en la ROM del floppy. Dichas órdenes no son válidas para usarlas con el casete, puesto que para acceder a ellas la unidad de disco debe encontrarse encendida. En cuanto a su sintaxis, todas ellas van precedidas por la barra vertical que se utiliza para cargar el CPM (|), obtenida mediante la pulsación simultánea de las teclas SHIFT 6 (&) y ARROBA (@). A continuación enunciaremos dichas órdenes:

|a
|b
|dir
|disc
|disc. in
|disc. out
|drive
|era
|ren
|tape
|tape.in
|tape.out
|user

Una vez enumeradas todas las órdenes específicas para disco, disponibles desde el lenguaje BASIC, pasaremos a describirlas detalladamente, no sin antes advertir de la necesidad de encender la unidad de disco antes que la unidad central, en el caso de que trabajemos con CPC 464 ó 472, o bien que dispongamos de una segunda unidad para el CPC6128. Este orden en la operación de arranque es necesario, dado que en el momento de la puesta en marcha de la unidad central, el ordenador efectúa un reconocimiento de los periféricos conectados, no siendo reconocidos aquellos que no estén encendidos.

Esta medida se hace estrictamente necesaria si vamos a trabajar desde CP/M y pasa a ser precautoria o meramente recomendable si simplemente nos disponemos a manejar la unidad de disco adicional desde el BASIC.

LOAD

Este comando debe tener la estructura sintáctica LOAD «nombre del fichero», por la cual cargará programas en BASIC, o código máquina.

El ordenador AMSTRAD reconoce distintos tipos de ficheros; así pues, se precisa de los mismos para distinguir programas BASIC de programas en lenguaje máquina, o para poder diferenciar programas de ficheros en general, como por ejemplo, un fichero de agenda. Esta distinción también la puede llevar a cabo la unidad de disco de modo interno, sin tener en cuenta el tipo de fichero indicado en el nombre de fichero.

Por tanto, llegamos a la conclusión de que el fichero .BAS deberá ser un programa en BASIC, el siguiente tipo, llamado .BIN, será un programa en código máquina, y por último, y para denotar ficheros secuenciales, se dispone de la extensión .SEQ.

Cuando se introduzca el comando LOAD, AMSDOS cargará el fichero «nombre de fichero...». En el caso de que dicho fichero no existiera, se intentará cargar el fichero «nombre de fichero.BAS», y si tampoco se encontrara este último, buscaría el fichero «nombre de fichero.BIN». Finalmente, si tampoco lograra localizarlo, aparecería en la pantalla el siguiente mensaje: *file not found,* cuya traducción al castellano es Fichero no encontrado.

Este sistema de búsqueda por extensiones (ninguna, .BAS y .BIN) es posible obviarlo indicando directamente en el nombre de la extensión a emplear; método absolutamente necesario si el fichero a cargar carece de ninguna de las extensiones mencionadas (ninguna, .BAS y .BIN), o bien siempre que dado un mismo nombre de fichero con dos o tres extensiones de las indicadas, deseemos cargar alguna en concreto que altera el orden habitual de búsqueda; por ejemplo, un Nombre.BIN, cuando existe también un Nombre.BAS.

```
LOAD "nombre del fichero.BAS"
LOAD "nombre del fichero.BIN"
```

Para poder cargar un programa o fichero de la segunda unidad de disco, debemos incluir en la orden el indicativo de dicha unidad, quedando el comando con el siguiente aspecto:

```
LOAD "B: nombre del fichero"
```

Como es lógico, en el caso de la unidad B, también podemos indicar la extensión del fichero a cargar.

Si la posición del disco no fuera la correcta, por estar incorrectamente insertado o ni siquiera introducido, aparecería el siguiente mensaje de error en la pantalla:

Drive A: disc missing
Retry, Ignore or Cancel?

cuya traducción es

Unidad A: sin disco
Reintentar, Ignorar o Cancelar?

Pulsaríamos pues, la tecla «R» (por RETRY), es decir reintentar. En el caso de que aparezca el mensaje de error:

Drive A: read fail
Retry, Ignore or Cancel?

de significado

Unidad A: error de lectura
Reintentar, Ignorar, Cancelar?

sería señal de un error de lectura en el disco, motivado ya sea porque el soporte esté defectuoso o porque no se encuentre correctamente formateado.

RUN

Este comando se corresponde con todas las características sintácticas del comando LOAD anteriormente descrito.

Sin embargo, existe una pequeña pero sustancial diferencia entre ambos. El comando RUN ejecuta automáticamente el programa una vez que éste ha sido cargado.

Los mensajes de error que se pueden presentar con este comando son los mismos que para LOAD.

CHAIN

El comando CHAIN se encarga de leer el programa del disco y cargarlo en la memoria, reemplazando al actual y ejecutando el nuevo programa, bien desde el principio, o bien desde una línea especificada como parámetro opcional.

CHAIN "nombre del fichero.BAS",220

MERGE

Su función es leer el programa indicado y superponerlo al almace-

nado en la memoria, sustituyendo las líneas de igual número que existieran en el antiguo.

MERGE "nombre del fichero.BAS"

Como tendremos ocasión de comprobar, esta instrucción es particularmente útil en trabajos de programación.

CHAIN MERGE

De su propio nombre, es fácil deducir que este comando combina los dos descritos anteriormente. Así pues, por una parte realiza la función MERGE, superponiendo el nuevo programa leído al anterior, sustituyendo las líneas de igual numeración, y por otro lado, realiza la función CHAIN de ejecutar el programa, bien desde el principio o bien desde la línea que indiquemos en el parámetro opcional.

CHAIN MERGE "nombre del fichero.BAS",580

CAT

Mediante esta orden obtendremos información acerca de los ficheros que se encontrasen en nuestro disco, y de la memoria de la que disponemos en el mismo.

La orden CAT nos da la longitud de los ficheros en KBytes, ordenándolos alfabéticamente antes de darles salida (la mínima unidad para un fichero es un KByte).

En caso de no especificarlo, la orden nos dará el índice de los ficheros almacenados en el disco bajo el número de usuario (USER) 0. Para poder consultar los ficheros almacenados bajo otro número de USER, deberemos incorporar dicho número a la orden CAT, aunque esto será tratado más adelante.

CLOSEIN CLOSEOUT

El primero de los comandos, cierra el fichero del disco que esté abierto en dirección de entrada, entendiéndose por «dirección de entrada», el tipo de fichero que ha sido abierto con el fin de leer sus datos. CLOSEOUT cierra el fichero del disco que se encontrase abierto en dirección de salida, es decir, que fue abierto para escribir datos en él.

OPENIN Y OPENOUT

La función de CLOSEIN es abrir en dirección de entrada, un fichero grabado en disco. Tiene por lo tanto el objetivo inverso de la orden CLOSEIN. En contraposición a OPENIN, abre en dirección de salida, un fichero para su grabación en disco.

PRINT#9

Mediante el comando PRINT#, podemos referirnos a una de las ventanas definidas, con n comprendida entre 0 y 7, con n=8 a la impresora y con n=9 a la unidad de disco.

Con la orden PRINT#9 escribimos datos en disco, de forma similar a como lo haríamos en la pantalla si no especificáramos ningún canal (#9); por tanto, para el empleo de esta orden debe existir algún fichero abierto en dirección de salida. Este comando, cuando va seguido de una coma, separa la línea, mientras que si colocásemos un punto y coma al final de la orden PRINT, la siguiente salida sería escrita en la misma línea. En salidas de disco podemos crear también un fichero por encadenamiento de varias salidas.

INPUT#9

Si con la orden PRINT#9 escribíamos datos en el disco, con INPUT#9 podemos recuperarlos.

Al igual que si tomáramos datos desde el teclado, la orden debe ir seguida por un nombre de variable, la cual contendrá el dato leído.

El separador empleado puede ser un punto y coma o una coma, dando lugar el primer tipo a que se escriba el signo de interrogación, sirviendo la coma para impedirlo.

LINE INPUT#9

La función de este comando es también la lectura de datos.

El separador deberá ser también un punto y coma o una coma. El punto y coma haría que se escribiese el signo de interrogación y la coma lo impediría, como en el caso del INPUT#9.

Como vamos viendo hasta ahora, las características de las funcio-

nes INPUT#9 Y LINE INPUT#9 son idénticas, pero no en vano podemos destacar algunas diferencias.

Sabemos que los campos de datos se encuentran separados entre sí por símbolos separadores. Estos símbolos separadores son el punto y coma, la coma, el retorno del carro y el carácter EOF (del que hablaremos más adelante).

La diferencia entre ambas órdenes estriba, obviamente, en algo que pueda hacer el segundo de los comandos que no pueda cumplir el primero. En concreto, se trata de algo directamente relacionado con los datos que es posible leer.

No podremos recuperar mediante INPUT#9 una cadena que contenga una coma (a menos que encerrásemos la cadena entre comillas), ya que la coma sería interpretada como símbolo separador. Pero las comillas no pueden ser leídas de una cadena por el comando INPUT. Sin embargo, LINE INPUT lee cualquier carácter, siendo solamente válido el retorno del carro como separador, por lo que acepta también comas y comillas.

LIST#9

Mediante este comando, almacenamos en disco el listado del programa presente en la memoria. Podemos incorporar el número de la línea inicial a partir de la cual queremos que se almacene en el disco, y señalar igualmente el número de la última línea a grabar.

En un principio esta instrucción puede parecer igual a SAVE. La diferencia entre ellas estriba en que mientras SAVE almacena las palabras clave (ej. DATA) en forma de «tokens», es decir, codificadas, la orden LIST#9 almacena las palabras claves letra a letra, como si se tratara de cualquier otro literal.

Recuperaremos el listado grabado en el disco con esta orden mediante la instrucción INPUT o LINE INPUT.

SAVE

Con el comando SAVE grabaremos en disco el programa residente en la memoria.

Al programa grabado mediante esta instrucción le podremos dar nombre con una cadena de ocho caracteres como máximo. Podemos incorporar, asimismo, una extensión de tres caracteres, generando

una denominación final de 11 caracteres, separados los tres últimos de los ocho primeros por un punto.

SAVE "nombre.ext"

En caso de querer grabar un programa en estado de protección, añadiremos una «P» a la instrucción.

SAVE "nombre.ext",P

De esta forma, el programa ya no podrá ser listado, reinicializado o almacenado tras haber provocado una interrupción (ESC).

También podemos grabar el fichero en modo ASCII, es decir, como si se efectuara un listado, con sus palabras clave *(token)* expandidas letra por letra.

SAVE "nombre.ext",A

Por otra parte, es posible efectuar la grabación de programas en lenguaje máquina o ficheros binarios, cuya extensión es .BIN. En este caso, hay que incluir indicaciones acerca de la dirección inicial y la longitud del fichero a grabar.

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

Así por ejemplo, la memoria dedicada a la pantalla podremos grabarla en forma de fichero binario. Esto es lo que se denomina «volcado de pantalla».

Para el caso de querer volcar la pantalla la instrucción tendría el siguiente aspecto.

SAVE "pantalla",B,&C000,&4000

La orden SAVE al igual que LOAD, podrá incorporar al fichero, indicaciones acerca de la unidad de disco y número de USER.

EOF

Mediante este comando, podemos consultar y por lo tanto saber si hemos alcanzado el final de un fichero.

La marca de EOF *(End Of File,* final de fichero), es situada por el sistema operativo cuando cerramos un fichero mediante la orden CLOSEOUT.

EOF se utiliza como si se tratara de una variable cualquiera. De esta forma, si EOF da valor −1, implica que no hay ningún fichero abierto o que hemos alcanzado el final; por el contrario, tendrá valor 0 en caso de no haber alcanzado el final del fichero.

Construiremos bucles de lectura mediante las instrucciones WHILE y WEND.

WRITE#9

Este comando se puede considerar análogo a PRINT#9; al igual que éste, su función consiste en direccionar hacia el disco datos contenidos en constantes o variables, ya sean numéricas o cadenas de caracteres.

Lo que diferencia un comando del otro, son los caracteres separadores. Mientras que la instrucción PRINT considera como caracteres separadores el retorno de carro y la coma, para WRITE el único carácter separador es el retorno de carro.

Así por ejemplo, si introducimos el siguiente listado:

```
 10 OPENOUT "Ejemplo"
 20 PRINT #9,"Fernandez, Antonio"
 30 PRINT #9,"Perez, Amalia"
 40 CLOSEOUT
 50 OPENIN "Ejemplo"
 60 INPUT #9,PERSONA1$
 70 INPUT #9,persona2$
 80 CLOSEIN
 90 PRINT "persona1 = "persona1$
100 PRINT "persona2= "persona2$
110 END
```

Lo que esperamos que el ordenador escriba al ejecutar el programa es lo siguiente:

```
persona1 = Fernández, Antonio
persona2 = Pérez, Amalia
```

Pero, sin embargo, aparecerá:

```
persona1 = Fernández
persona2= Antonio
```

Ello es debido a que la orden PRINT ha interpretado la coma como un carácter separador.

Para solventar este problema sustituiremos PRINT por WRITE, y conseguiremos que todo lo que se encuentre encerrado entre las comillas sea considerado como elemento de la cadena, dejando únicamente fuera de ella las comillas.

Si a continuación introducimos la siguiente línea

```
WRITE 10,11,"prueba, probando.",12
```

en la pantalla aparecerá lo siguiente tras su lectura

```
10,11,"prueba, probando.",12
```

Sin embargo, si lo hubiéramos hecho mediante el comando PRINT, el resultado sería este otro:

```
10  11  prueba, probando.  12
```

En este caso, las comas han provocado un salto del tabulador, dejando espacios vacíos.

Como quiera que en el disco también se almacenan los espacios vacíos, este sistema nos llevará a un consumo de memoria superior al caso de haber utilizado WRITE. Por lo tanto, WRITE está indicado para que las comas sean reconocidas como tales, con un ahorro de memoria que nunca viene mal.

COMANDOS EXTERNOS

omo complemento a las órdenes residentes del AMSDOS, existe un segundo juego de comandos, denominados «externos». Dichos comandos se encuentran almacenados en una ROM, localizada en la unidad de disco. Este es el motivo por el cual estos comandos reciben el nombre de «externos».

Las órdenes externas son un total de catorce y al contrario de lo que sucedía con las órdenes residentes, que son también válidas para el casete, las órdenes externas son específicas de la unidad de disco.

Todas estas órdenes van precedidas de la barra vertical (|). La lista completa de dichas órdenes, es la siguiente:

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

Ni que decir tiene que para acceder a las órdenes externas, es necesario que la unidad de disco se encienda antes que la unidad central, en el caso de que dicha unidad no se encuentre incorporada.

|A

Este comando determina la unidad de disco A como unidad de disco estándar; entendiendo como tal a la que van referidas las órdenes de floppy sin ningún tipo de especificación suplementaria. El uso de esta orden, no tiene sentido en el caso de disponer solamente de una unidad de disco.

Esta orden tiene efecto idéntico a:

```
a$="a"
|DRIVE,@a$
```

que más adelante estudiaremos.

|B

Al igual que la orden |A, determina o fija la unidad de disco B como unidad de disco estándar (anteriormente descrita).

Es en todo equivalente a:

```
a$="B"
|DRIVE,@a$
```

|CPM

Mediante este comando cargaremos el sistema operativo CP/M del disco. Así pues, al ejecutarse, en la unidad de disco A deberá encontrarse un disco que haya sido formateado con CP/M en ambas pistas del sistema (formato System de Diskit3), y en el cual se halle el fichero de arranque CP/M, de extensión .EMS *(Early Morning Start-up)*.

|DIR

Mediante este comando podremos saber el contenido del disco,

mostrándose el directorio del mismo, así como la memoria de que disponemos en el soporte. Dicho directorio se presentará según la norma CP/M (ésta es su diferencia con CAT), lo cual supone que no se especificará la ocupación de cada fichero, ni se mostrarán clasificados por orden alfabético.

El comando puede hallarse opcionalmente seguido por una expresión de cadena, separada de la orden por una coma, en la que se indica el tipo de ficheros a mostrar. En dicha expresión se puede emplear los caracteres comodín interrogación (?) y asterisco (*), de uso habitual en CP/M. En el caso de que la expresión no se indique, se le asume el valor por defecto «*.*» (todos los ficheros).

Así por ejemplo, si deseáramos listar todos los ficheros BASIC (de extensión .BAS) emplearíamos la orden

```
|DIR,"*.BAS"
```

la cual es enteramente equivalente a

```
a$="*.BAS"
|DIR,@a$
```

Por el contrario, si el capricho nos llevara a querer averiguar la presencia en el disco, de ficheros cuyo nombre comience por GBA y extensión comience por I y acabe por G, utilizaríamos la fórmula

```
|DIR,"gba*.i?g"
```

o bien

```
a$="gba*.i?g"
|DIR,@a$
```

Como vemos, la ventaja esencial de CAT sobre DIR estriba en el ordenamiento alfabético del directorio y la indicación de las longitudes individuales de cada fichero, mientras que DIR facilita la posibilidad de mostrar en el directorio sólo los ficheros que se correspondan con un determinado tipo.

|DISC

Mediante esta orden, se designa al disco como periférico en el que se efectúan por defecto todas las operaciones de lectura y escritura de datos.

Esta orden reúne en un solo comando los efectos de |DISC.IN y |DISC.OUT.

|DISC.IN Y |DISC.OUT

El primero de los comandos designa al disco como periférico en el que se efectuarán por defecto las lecturas de datos. |DISC.OUT, por contra, como habremos imaginado, determina que las escrituras de datos se efectúen por defecto en la unidad de disco.

Obviamente, estas dos órdenes, así como |DISC, carecen de gran utilidad en el CPC 6128, a no ser que empleemos con frecuencia un casete externo como medio de almacenamiento masivo.

|DRIVE

Esta orden realiza la misma función que los comandos |A y |B. La diferencia radica en que |DRIVE es flexible, puesto que puede indicar, mediante una expresión de cadena, cuál es la unidad de disco elegida como estándar o implícita.

```
a$="b"
|DRIVE,@$
```

A partir del momento en que se ejecutara esta orden, la unidad B quedaría como unidad estándar 9por defecto). El uso de este comando no tiene sentido alguno en el caso de disponer solamente de una unidad de disco y por tanto, será inoperante cuando se consiga detectar la presencia de la unidad especificada.

|ERA

Cuando necesitemos borrar del disco algún fichero almacenado en el mismo, |ERA es la orden a la cual deberemos recurrir; para ello, especificaremos tanto el nombre como la extensión del fichero a borrar, dado que por defecto no presupone ningún tipo de extensión específica, salvo la vacía.

```
|ERA,"nombre.ext"
```

En esta orden también está permitido el uso de comodines, aunque dada su delicada e «irreparable» misión, es conveniente manejarlos con sumo cuidado. Así por ejemplo, si queremos borrar todos los ficheros de extensión .BAK (copias de seguridad de ficheros ya existentes y actualizados) introduciremos la siguiente línea

```
|ERA,"*.BAK"
```

Asimismo, podemos indicar el nombre del fichero utilizando variables de cadena.

```
a$="*.SEQ"
|ERA,@a$
```

Al ejecutar estas líneas se borrarían todos los ficheros de tipo secuencial.

```
|ERA,"*.*"
```

Este comando, por contra, borraría todos los ficheros existentes en el disco.

|REN

Esta nueva orden externa nos permitirá cambiar el nombre del fichero que deseemos, ya almacenado en el disco, sin necesidad de volver a cargar y grabarlo nuevamente.

En su sintaxis, la orden va seguida de dos expresiones de cadena, separadas entre sí y del comando por comas, que indicarán, respectivamente, la denominación actual del fichero a cambiar y su nuevo nombre.

```
|REN,"nombre-nuevo.ext.","nombre-antiguo.ext"
```

Así por ejemplo, si queremos cambiar el fichero con el nombre de «Luis» por el nombre de «Antonio», introduciríamos la siguiente orden:

```
|REN,"Antonio.ext","Luis.ext"
```

Por otra parte, las expresiones de cadena a emplear pueden ser variables y no constantes, aunque en este caso habrán de ir precedidas del signo arroba (@).

```
ant$= "Luis.ext"
nue$= "Antonio.ext"
|REN,@nue$,@ant$
```

En lo referente al tratamiento de errores, hay que decir que en caso de intentar renombrar un fichero inexistente, aparecería el mensaje *nombre de fichero not found* (fichero no encontrado). Análogamente, la duplicidad de nombres se evita por el sistema, emitiendo el mensaje *nombre de fichero already exists* (fichero ya existente), en caso de que a cualquier fichero pretendiéramos otorgarle un nombre ya presente en el directorio.

Si bien no es admisible en |REN el uso de comodines, sí lo es la inclusión del parámetro USER. Así, si quisiéramos cambiar el nombre de un fichero USER 2, para que además pasara con la denominación cambiada al área USER 0, introduciríamos la siguiente línea.

```
|REN,"0;nuevo.ext.","2:antiguo.ext"
```

|TAPE

Este comando cumple la función inversa a la orden |DISC, anulando por tanto el efecto de la misma, y por supuesto, de sus correspondientes |DISC.IN y |DISC.OUT. Establece que las entradas y salidas deben dirigirse a la cinta. De forma similar a |DISC, |TAPE tiene dos órdenes asociadas cuyos efectos conjuga: |TAPE.IN y |TAPE.OUT.

|TAPE.IN Y |TAPE.OUT

La primera de estas órdenes desactiva los comandos |DISC y |DISC.IN, utilizándose a partir de su ejecución la unidad de casete para los ficheros de entrada.

Por contra, |TAPE.OUT inhibe las funciones de |DISC y |DISC.OUT, pero hace que se utilice la unidad de cinta como periférico de salida de ficheros.

De la presencia individual de estas dos órdenes, se desprende que es posible la combinación de los ficheros de entrada y salida del casete y del disco. Así, podremos leer datos en el disco, escribirlos en el casete y viceversa. Para ello, debemos proceder de la siguiente manera:

```
|TAPE
|DISC.IN
OPENIN "fichero"
OPENOUT "copia seguridad"
```

Con estas líneas conseguimos abrir un canal de entrada del disco para la lectura, y uno de salida hacia el casete para la escritura de los datos.

|USER

Con este comando es posible especificar una zona determinada de usuario (USER) dentro del directorio como área de actuación por defecto de ciertos comandos (LOAD, CAT, |DIR...). El número de usuarios que es posible establecer en el directorio alcanza un máximo de 16.

Mediante esta función es posible ocultar ciertos ficheros a ojos indiscretos, albergándolos en una determinada zona de usuario que sólo nosotros conozcamos. Si por ejemplo, teniendo un programa en memoria, ejecutamos las siguientes líneas

```
|USER,3
SAVE "programa"
|USER,0
CAT
```

No aparecerá en el índice el programa que acabamos de grabar. Si deseáramos recuperarlo o simplemente incluirlo en el directorio, deberíamos volver a situarnos en la zona de usuario en la que fue grabado; por ejemplo, con las líneas siguientes:

```
|USER,3
CAT
|USER,0
```

En esta ocasión aparecerá el índice de los programas que hayamos grabado bajo el USER 3. Dado que disponemos de 16 zonas de usuario, numeradas de la 0 a la 15, tendremos bastantes sitios donde proteger programas y ficheros de otros usuarios.

Por otra parte, las tres líneas que empleamos anteriormente para grabar un programa o fichero en una zona de USER determinada, pueden sustituirse por:

SAVE "3:programa"

del mismo modo que, como ya hemos visto, es posible cambiar ficheros de una zona de USER a otra mediante el comando |REN, sin necesidad de cargarlos y volver a grabarlos.

FICHEROS: SECUENCIALES, ALEATORIOS E INDEXADOS

En este capítulo, dedicado a los tipos de ficheros, empezaremos por definir este concepto.

Un FICHERO es una zona del disco que referenciamos mediante un nombre, encargada de almacenar información del más diverso tipo. Está formado por una serie de unidades más pequeñas, denominadas registros, los cuales a su vez se descomponen en campos, que son la mínima cantidad de información con sentido a la que se puede acceder.

Lo mejor en estos casos es fijarnos en un ejemplo concreto. Si consideramos la lista de los alumnos de un colegio, el nombre de pila de uno de ellos, sería un campo, el conjunto de número de matrícula, más nombre, más apellidos, más dirección, más curso que estudia, más la nota media, por ejemplo, un registro; y la suma de todos los alumnos anotados, además de los espacios todavía sin ocupar en cada curso, es decir, la propia lista, es lo que denominamos FICHERO.

Una vez visto ya lo que es un fichero, pasemos a definir los diversos tipos existentes.

FICHERO SECUENCIAL

La unidad de disco no solamente nos sirve para el almacenamiento de datos y carga de programas, sino también para el almacenamiento de grandes cantidades de datos, que no podríamos manejar, o al menos no sería muy difícil, con la unidad de casete. El almacenamiento secuencial de datos, no es el tipo de disposición de la información más rápido, pero sí el más sencillo para resolver problemas de una magnitud de mediana a pequeña.

Existen ficheros que contienen programas y otros que contienen datos, y aunque ambos son almacenados de modo secuencial, destacaremos entre ellos una gran diferencia:

Podremos leer los FICHEROS DE PROGRAMAS (BASIC-CODIGO MAQUINA) mediante el comando LOAD directamente de la memoria, mientras que los FICHEROS DE DATOS, sólo se podrán leer en memoria y gestionarse a través de programas.

Pongamos por ejemplo un programa de «química», sería lo que denominamos un FICHERO DE PROGRAMA, mientras que los datos obtenidos por este programa, se encontrarían almacenados en el disco dentro de un FICHERO DE DATOS.

Un fichero secuencial no es otra cosa que una secuencia o serie de caracteres. Si en un fichero secuencial quisiéramos, por ejemplo, leer el último carácter, tendríamos que leer previamente todos los anteriores; de ahí su denominación.

Este modo, como dijimos anteriormente, no es el más eficaz, pero por suerte la unidad de disco de los AMSTRAD es muy rápida, y por consiguiente nuestra lectura mediante un fichero secuencial no se verá afectada por este tipo de cosas, siempre y cuando no acometamos el tratamiento de grandes masas de datos.

Existe otra posibilidad de leer datos muchísimo más cómoda, accediendo directamente al fichero, se trata del acceso directo, también llamado relativo o aleatorio (RANDOM ACCESS).

Pondremos un ejemplo de su funcionamiento. Supongamos que tuviéramos un libro de 100 páginas. Por el sistema secuencial, si quisiéramos leer la página 89, deberíamos haber leído desde la página 1 a la 88. Mediante el acceso directo, podremos leer, por ejemplo, la página 50, sin ninguna necesidad de haber leído todas las anteriores.

Por otra parte, la lectura secuencial no permite en ningún momento el retroceso en la lectura, ni tan siquiera de modo secuencial, mientras que en los ficheros relativos es posible ir «saltando» de registro en registros, sin necesidad de clasificar previamente su lectura por orden ascendente.

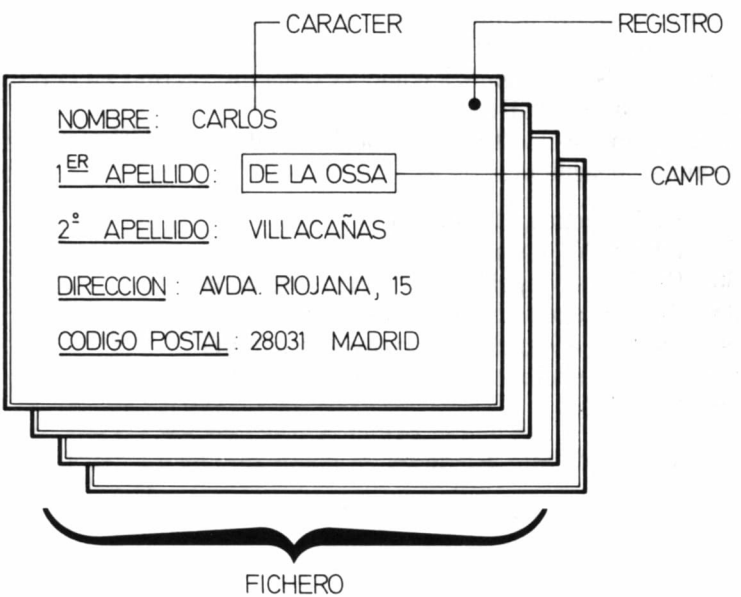
No obstante, más adelante entraremos en los detalles de estos ficheros, de forma que continuemos ahora con los secuenciales.

En el almacenamiento secuencial de datos, los distintos campos de datos se separan mediante un retorno de carro en el disco, así que mediante la tecla ENTER o RETURN indicaremos la finalización de un campo de datos.

Dichos campos no es imprescindible que tengan la misma longitud, lo cual reporta una indudable ventaja de flexibilidad frente a otros sistemas.

Para leer un fichero de este tipo, utilizaremos el comando INPUT #9. Por ejemplo:

```
OPENIN"nombre del fichero"
INPUT#9,campo1$,campo2$,campo3
CLOSEIN
```



20.3.1. Disposición de la información en un fichero.

CÓMO SE PROGRAMAN LOS FICHEROS SECUENCIALES

El listado que se encuentra a continuación es un ejemplo de la forma más simple de crear un fichero secuencial.

```
10  REM fichero secuencial
20  OPENOUT "datos.seq"
30  REM primer registro
40  PRINT #9,"González"
50  PRINT #9,"Luis"
60  PRINT #9,"9597867"
70  REM segundo registro
80  PRINT #9,"Jimenez"
90  PRINT #9,"Jose"
100 PRINT #9,"9598767"
110 REM fin del fichero
120 CLOSEOUT
130 END
```

Con este listado hemos creado un fichero secuencial llamado «datos.seq» que contiene dos registros o conjuntos de datos (campos). El comando PRINT#9 significa como ya sabemos, escribir en el disco, y se usa como un PRINT normal.

Para recuperar nuestros datos del disco, debemos introducir un listado similar al siguiente:

```
10  REM lectura del fichero
20  OPENIN "datos.seq"
30  INPUT #9,apellido$,nombre$,telefono$
40  PRINT apellido$,nombre$,telefono$
50  INPUT #9,apellido$,nombre$,telefono$
60  PRINT apellido$,nombre$,telefono$
70  CLOSEIN
```

El resultado de este listado será:

```
González        Luis        9597867
Jimenez         José        9598767
```

En este listado hemos utilizado el comando INPUT#9, el cual lee datos del disco. Este pequeño programa ha funcionado a la perfección, pero en caso de tener que almacenar gran cantidad de juegos de

datos, no deja de ser engorroso. Para ahorrarnos trabajo, lo más sencillo es incluir un bucle en el programa.

```
10  REM lectura con bucle
20  OPENIN "datos.seq"
30  FOR I=1 TO 2
40  INPUT #9,apellido$,nombre$,telefono$
50  PRINT apellido$,nombre$,telefono$
60  NEXT I
70  CLOSEIN
80  END
```

Este listado puede tener la variante de leer un solo dato del disco y escribirlo en la pantalla, para seguir con el siguiente, etc.

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

Tras la lectura de un registro,el puntero indica dónde debe continuar leyendo el siguiente, pero si ya hemos leído el último dato de fichero y pretendemos leer otro más, como el puntero estará señalando a la marca EOF *(End Of File,* fin de fichero), obtendremos el mensaje de error «EOF met» *(End Of File met,* fin de fichero encontrado).

Para solventar este inconveniente, debemos comprobar si hemos alcanzado el final del fichero mediante el comando BASIC «WHILE-WEND».

```
10  Lectura de datos
20  OPENIN "datos.seq"
30  WHILE NOT EOF
40  INPUT #9,apellido$,nombre$,telefono$
50  PRINT apellido$,nombre$,telefono$
60  WEND
70  CLOSEIN
80  END
```

Este listado es básicamente correcto. Pero tiene un pequeño inconveniente. En caso de que alguno de los juegos de datos no tenga

tres campos, daría al traste con la lectura. Para solucionar este problema, hagamos la siguiente modificación:

```
10 REM lectura mejorada
20 OPENIN "datos.seq"
30 WHILE NOT EOF
40 INPUT#9,dato$
50 PRINT dato$
60 WEND
70 CLOSEIN
80 END
```

Hasta aquí todo perfecto, pero estos cortos programas no sirven para modificar un campo de datos y almacenarlo. Para este fin, debemos utilizar una variable suscrita.

```
10 DIM apellido$(2),nombre$(2),telefono$(2)
20 REM area de lectura
30 OPENIN "datos.seq"
40 a=0
50 WHILE NOT EOF
60 a=a+1
70 INPUT#9,apellido$(a),nombre$(a),telefono$(a)
80 WEND
90 CLOSEIN
100 END;REM aquí comenzaría el tratamiento de los datos
```

En la línea 10 debemos definir el número límite del índice correspondiente al máximo número de datos (en nuestro ejemplo 2), o bien, en caso de desconocer a priori el número de datos, a un cierto valor que se considere nunca se llegará a alcanzar, siempre dentro de los límites de la memoria de nuestro ordenador.

Un programa de este estilo nos permite la lectura de ficheros a través de una variable suscrita, que conserva en cierto modo la estructura secuencial de los datos leídos, motivo por el cual, siempre que la memoria disponible lo permita, es muy aconsejable utilizar este sistema; una vez modificados los datos en las propias variables, podremos efectuar la grabación del fichero con un programa prácticamente idéntico, sólo diferenciado del anterior en el sentido del trasvase de datos.

```
200 REM área de grabación de datos
210 REM suponemos que el tratamiento de los datos
220 REM se efectuo a partir de 100
```

```
230 OPENOUT "datos.seq"
240 REM bucle de escritura
250 FOR i=1 TO a-1
260 PRINT#9,apellido$(i),nombre$(i),teléfono$(i)
270 NEXT
280 CLOSEOUT
290 END
```

En este caso no ha sido preciso el control de la variable EOF, motivo por el cual hemos podido emplear una estructura de bucle FOR-NEXT; por otra parte, recordemos que la variable A es la que contiene el registro siguiente a leer, debido a lo cual el número de registros leídos se establece en a-1.

FICHERO RELATIVO O ALEATORIO

Habiendo ya tratado sobre el almacenamiento secuencial de datos, pasaremos a describir el almacenamiento relativo de datos, del cual hemos recibido ya unas breves nociones.

En el sistema operativo de Amstrad no está previsto el almacenamiento relativo de datos, y ello significa que no existen órdenes para su tratamiento. Por lo tanto, para explicar dicha forma de almacenamiento de datos nos basaremos en ejemplos.

En la gestión relativa de datos, un fichero se abre siempre para la lectura y escritura indistintamente. Además, al contrario de lo que ocurre con el almacenamiento secuencial de datos, hemos de olvidarnos de que un registro sigue al otro, accediendo en el almacenamiento relativo a cualquier registro del fichero, cualquiera que sea su posición.

No obstante, el nuevo sistema conlleva ciertas restricciones y exigencias. Para comenzar, refrescaremos el concepto de puntero de datos. Como recordaremos de la explicación del funcionamiento secuencial, un puntero de datos recorría el fichero SECUENCIALMENTE, desde el primer registro, campo por campo, leyéndolos todos hasta localizar el que deseábamos o llegar al final del fichero.

La adopción de un sistema de acceso directo, no conlleva la desaparición de dicho puntero, dado que seguimos precisando de un elemento que nos permita señalar qué campo deseamos leer. Lo que sucede es simplemente que el puntero de los ficheros aleatorios podemos moverlo libremente por los registros del fichero, saltando de uno a otro directamente, sin necesidad de pasar por los anteriores.

¿Cómo indicamos el registro que deseamos? Nada más fácil, sim-

plemente indicamos su número s de orden dentro de los que componen el fichero, tomando como punto de referencia el inicio del mismo. Así por ejemplo, si tras la apertura de un fichero relativo requerimos la lectura de su vigésimo registro, el puntero sabrá que lo único que tiene que hacer para situarse correctamente, es desplazarse 20 registros más adelante. Si ahora reclamáramos la información del decimosexto registro, el puntero simplemente tendría que retroceder cuatro, y así sucesivamente.

De esta explicación deducimos rápidamente porqué un casete nunca puede soportar ficheros relativos: una vez que se aprieta el botón de PLAY, no existe la posibilidad de alterar la dirección de la marcha de la cinta (retroceder).

No obstante, si bien la eficacia de los ficheros relativos frente a los secuenciales se hace patente, ya hemos anticipado que presenta ciertas restricciones. Sin duda, la más importante es la necesidad de una longitud de registro constante, igual para todos los integrantes del fichero; de no ser así, el puntero no sabría cuántos espacios avanzar o retroceder para acceder al próximo registro.

Así pues, en contraposición a los secuenciales, los ficheros directos adolecen de una cierta falta de flexibilidad en cuanto a la longitud de sus registros.

De modo distinto a lo que ocurría con el fichero secuencial, en los relativos hay que disponer primeramente un fichero, antes de que pueda ser escrito.

Debemos pues, hacernos una idea exacta del tamaño de cada registro, previamente a la creación de un fichero relativo. De hecho, deberemos tener en todo momento la precaución de no sobrepasar dicha longitud de registro, dado que de no ser así, los datos de uno sobreescribirían los del siguiente.

Dado que antes de la creación de cualquier fichero aleatorio es preciso saber la longitud máxima de los registros que lo integran, comencemos por este punto, y para ello nada mejor que un ejemplo. Supongamos que queremos crear una agenda telefónica, distribuyendo los campos de datos de la siguiente manera:

Nombre-apellidos	45 caracteres
Calle	30 caracteres
Provincia	22 caracteres
Teléfono	15 caracteres
Distrito postal	12 caracteres
Total	124 caracteres

Cada registro comprende por tanto 124 caracteres. Una vez hayamos calculado la longitud de nuestro registro, deberemos estimar con gran precisión la cantidad de registros que contendrá nuestro fichero.

Por ejemplo, si contáramos con 50 fichas de la estructura anterior, y esperásemos que llegasen a duplicarse, deberíamos disponer del espacio para 100 registros. Otro sistema es pensar en la capacidad de un disco. Si por ejemplo, una vez descontado el espacio que ocupan los programas de gestión de nuestra agenda, nos quedaran en el diskette 130 K, sería fácil estimar que podemos tener un fichero tan grande como nos quepa en ese espacio. Dado que 1 K son 1.024 caracteres, 130 K permitirían la entrada de hasta 133.120 o lo que es lo mismo, 1.073 registros (133.120/124 = 1073.5483). Para redondear, fijemos definitivamente en 1.000 el número máximo de registros de nuestra agenda; ¡no está nada mal!

Por otra parte, no hemos de preocuparnos de que tal masa de datos no entre en la memoria de nuestro ordenador, dado que el acceso aleatorio va a permitirnos manejar ficheros aleatorios como si fuesen arrays BASIC (DIM), es decir, referenciando cada registro por su número de orden, debido a lo cual no es necesario más que leer un registro en cada ocasión.

En cuanto a la numeración de los registros, ésta comienza generalmente desde el 1 y sirve de índice de referencia para localizar cualquiera en el disco, basándose por supuesto en su longitud constante.

Una vez que hemos definido los tipos de fichero secuencial y relativo o aleatorio, daremos una serie de consejos para su mejor utilización.

Ya hemos visto que el almacenamiento relativo de datos es el más cómodo, pero no es el más aconsejable para utilizarlo en problemas de mediana o pequeña magnitud, dada su poca flexibilidad. Así por ejemplo, si quisiéramos gestionar en nuestro Amstrad pequeñas tablas de datos, el modo de almacenamiento más correcto y cómodo sería el secuencial.

La utilización del almacenamiento relativo de datos empieza a ser muy interesante y necesaria a partir de un cierto número de registros, por su lentitud de tratamiento, o bien cuando resulta imposible retener todos los datos en memoria.

Entre los programas de regalo que recibimos al adquirir un CPC 664 ó 6128, o al añadir al CPC 464 una unidad de disco, encontraremos uno de extraordinaria utilidad: RANDOM FILES.

Si traducimos este nombre al castellano, «Ficheros Aleatorios», inmediatamente podemos hacernos una idea de por dónde van los ti-

ros. Efectivamente, se trata de una serie de dos programas, RANDOM-F.BAS y RANDOM.BIN, encargados de dotar al sistema de disco de nuestro Amstrad de la posibilidad de crear y gestionar ficheros de acceso directo. ¿Qué significa esto?

Lo mejor será fijarnos en un ejemplo concreto; si consideramos la lista de los alumnos de un colegio, el nombre de pila de uno de ellos, sería un campo, el conjunto número de matrícula, más nombre, más apellidos, más dirección, más curso que estudia, más la nota media, por ejemplo, un registro; y la suma de todos los anotados, además de los espacios todavía sin ocupar en cada curso, es decir, la propia lista, es lo que denominamos fichero.

RANDOM.BIN

Esta formidable mini-rutina (ocupa tan sólo 1.792 bytes) amplía las posibilidades del Locomotive BASIC con cuatro nuevos mandatos: |OPEN, |CLOSE, |READ, y |WRITE.

Lo cierto es que se comporta estupendamente, siempre y cuando sepamos hacer buen uso de ella. Y la razón de esto, estriba en que sus creadores pusieron el máximo interés en gestionar ficheros aleatorios o relativos de una manera fácil, sin complicaciones, pero no hicieron mucho hincapié en los problemas que pudieran traer asociados los errores cometidos al programa.

Por ello, antes de ponerse a programar, es del todo imprescindible poner atención en determinados puntos:

1. Conocer a la perfección el significado de cada comando y los parámetros que lleva asociados.
2. Antes de ejecutar cualquier parte de un programa que haga uso de RANDOM.BIN, grabar en un disco, diferente al que contiene los ficheros aleatorios, la versión sin probar del programa.

Aunque parezca inocente este consejo, de no seguirlo, podemos enfrentarnos con degeneraciones en la información, tanto de los programas como los ficheros del disco, sin posibilidad de posterior solución.

Por tanto, ante todo «precaución», RANDOM.BIN funciona perfectamente, y está preparada para gestionar ficheros aleatorios, pero no para depurar los errores de programación.

SECUENCIAL Y ALEATORIO

Supongamos ahora que el colegio decide adquirir un Amstrad

CPC para informatizar la gestión de sus alumnos. Don Carlos, director del centro, que de esto de programar sabe un montón, decide crear un fichero secuencial en el CPC, asignando a cada alumno un número de matrícula. Sin embargo, al poco tiempo descubre alarmado que cada vez que desea realizar una consulta de cualquier dato referente a un alumno de número de matrícula alto, su ordenador se eterniza hasta que por fin da con él.

Una vez decidido por los ficheros relativos, en primer lugar se debe ejecutar el programa RANDOM-F (RUN «RANDOM-F») encargado de reservar el espacio necesario para la correcta utilización de la rutina.

RANDOM-FORMAT

Una vez ejecutado, se presenta una pantalla en la que se solicita la respuesta a las siguientes cuestiones:

NOMBRE DEL FICHERO
NUMERO DE FICHAS
LONGITUD DE FICHA
DISCO (A/B)

El nombre del fichero es el que nosotros deseemos asignarle, y como máximo contará con ocho caracteres de largo. Si continuamos con nuestro ejemplo, podemos denominarlo «alumnos».

Número de fichas (o de registros) es el máximo de anotaciones que pensamos realizar (sería el número de alumnos del colegio). Conviene sobredimensionar esta cantidad para evitar que en un momento determinado se nos pueda quedar pequeño el fichero.

Longitud de ficha es el número de caracteres que tendrá cada registro, es decir, la suma de los ocupados por cada uno de sus campos. Si por cada alumno, por ejemplo, queremos anotar:

— Nombre (12 caracteres).
— Apellidos (20 caracteres).
— Curso (2 caracteres).
— Nota media (4 caracteres, si consideramos que se puntúa sobre 10 máximo, y con dos decimales. Además, el punto decimal también ocupa una posición).

la longitud de cada registro es, por tanto, de 38 caracteres en total.

En este sentido conviene puntualizar que la mínima longitud de ficha definible es de 3 caracteres. Sin embargo, puede ser modificada actuando sobre las líneas 217 y 240 del listado del programa RANDOM-F.BAS.

Finalmente, la unidad de disco nos permite elegir entre la A o la B, como soporte del diskette que almacenará los ficheros definidos.

Tras ello, el programa RANDOM-F pide conformidad a los datos introducidos, e inmediatamente a continuación, procede a la creación del fichero en el disco, reservando el espacio necesario. Este tamaño viene dado por la fórmula:

$$EF(\text{en Kbytes}) = INT(NF \times LF = 1024)$$

siendo EF, el tamaño del fichero; NF, el número de fichas o registros; LF, la longitud de la ficha, el INT la parte entera (sin decimales) de la cantidad resultante.

En principio, no es posible tener un fichero con una capacidad superior a 150.000 bytes, auque sustituyendo en la línea 215 del programa dicha cantidad por otra mayor, será posible conseguirlo.

En realidad, el tamaño máximo de fichero lo limita la capacidad de disco en sí. Para trabajar con esta rutina de gestión de aleatorios, los discos utilizados como soporte de los ficheros deberán encontrarse en formato SISTEMA (no valen los formateados en DATA), y por tanto, con capacidad para albergar 169 Kbytes. Efectuando una sencilla operación:

$$169 \times 1.024 = 173.056 \text{ bytes}$$

concluimos que esta última cantidad, será el mayor tamaño posible a ocupar por un fichero manejado por RANDOM FILES.

TRABAJANDO CON RANDOM.BIN

En principio, para activar las cuatro nuevas órdenes para la gestión de ficheros aleatorios, es necesario introducir en la unidad el disco que contiene la rutina RANDOM FILES, y ejecutar las siguientes instrucciones:

```
MEMORY &9BFF
LOAD "RANDOM.BIN".&9C00
CALL &9C00
```

Lo primero que se debe hacer antes de acceder a los datos de un fichero es «abrirlo». Durante esta operación, el sistema operativo, reserva una zona de memoria intermedia denominada *buffer,* entre el disco y la memoria principal, utilizada como almacenamiento temporal de los datos transferidos en uno u otro sentido. RANDOM.BIN define como *buffer,* un área de 512 bytes.

Además, se informa de qué ficheros se desea mantener abiertos, su nombre, longitud de registro, y la unidad de disco donde el sistema los debe buscar. Para ello, disponemos del mandato OPEN. Su sintaxis completa es la siguiente:

```
|OPEN,@V$,NF,LR,UD
```

Analicemos detenidamente cada uno de los parámetros:

— V$ es una variable cualquiera alfanumérica en la que previamente habremos almacenado el nombre del fichero.

— NF es el número de fichero. Con esta rutina es posible mantener abiertos simultáneamente un máximo de 15 ficheros. Podemos por tanto, asignarle a cada fichero un número cualquiera entre 1 y 15, pero una vez seleccionado, en las operaciones de lectura o escritura, debemos identificar el fichero mediante el número elegido.

— LR es la longitud de registro en el fichero seleccionado. Naturalmente, debe coincidir con la definida anteriormente durante la creación del archivo, al ejecutar el programa RANDOM-F.BAS.

— UD es la unidad de disco en la cual se encontrará alojado el diskette que contiene al fichero en cuestión.

Suponiendo que nuestro fichero de alumnos se hallara en la unidad A, para abrirlo, sería necesario, por ejemplo, ejecutar la orden:

```
A$="alumnos":|OPEN,@A$,1,38,1
```

Escritura Y Lectura De Información

Para llevar a cabo estas operaciones disponemos de dos mandatos: WRITE y READ. Los dos siguen una construcción similar:

```
|WRITE,@V$,NR,NF      |READ,@V$,NR,NF
```

V$, al igual que en el mandato OPEN, es una variable alfanumérica cualquiera que previamente deberá ser dimensionada, pero en estos casos, su longitud ha de ser la misma que la del registro. Es muy importante no cometer errores en este punto concreto.

NR es el número de registro a leer o escribir. Naturalmente, debe estar comprendido entre 1, y el máximo seleccionado durante el programa de creación del fichero.

NF, como antes, es el número de fichero, que debe coincidir con el asignado en el mandato OPEN.

Continuando con nuestro ejemplo, supongamos que pretendemos escribir los datos de alumno con número de matrícula siete:

```
NOMBRE ..... MARIA JOSE
APELLIDOS ..... DEL MANZANO SERRANO
CURSO ..... 3C
NOTA MEDIA ..... 7.5
```

El proceso a seguir consistiría en crear una variable de 38 caracteres de largo, que contuviera cada dato en su lugar, para posteriormente, poder recuperarla del disco, e interpretarla. De la posición 1 a la 12 iría el nombre, de la 13 a la 32, los apellidos, en la 33 y 34, el curso, y finalmente, de la 35 a la 38, su nota media:

```
A$="MARIA×JOSE××DEL×MANZANO×SERRANO×3C7.5×"
```

Para evitar confusiones hemos rellenado los espacios en blanco con signos «×», pero, ¡representan blancos! Una vez construida la variable que contiene los datos a escribir en su lugar correspondiente ejecutaríamos la instrucción:

```
|WRITE,@A$,7,1
```

Si ahora lo que deseamos es leer los datos de nuestro primer alumno, el proceso a seguir es muy parecido. Lo primero, como siempre, dimensionar una variable del mismo tamaño que el registro a leer. Por ejemplo, podemos crear una variable L$ rellena en principio de blancos: L$=SPACE$(38).

A continuación llevamos a cabo, la operación de lectura en sí:

```
|READ,@L$,7,1
```

Tras ello, los datos de nuestra amiga María José quedarán almacenados en la variable L$, y tan sólo nos queda decodificarlos (interpretarlos). Para ello, disponemos de las funciones de manejo de cadenas del Locomotive BASIC, RIGHT$, MID$, y LEFT$. El aspecto de las instrucciones que llevarían a cabo esta operación, podría ser el siguiente:

```
100 PRINT "NOMBRE ..... "LEFT$(L$,12)
110 PRINT "APELLIDOS ..... "MID$(L$,13,20)
120 PRINT "CURSO ..... "MID$(L$,33,2)
130 PRINT "NOTA MEDIA ..... "RIGHT$(L$,4)
```

El último mandato que incorpora RANDOM.BIN es CLOSE. Su misión es contraria a la de OPEN, puesto que se encarga de «cerrar» el fichero. Aunque su importancia pueda parecer secundaria, es fundamental ejecutar este mandato cuando terminemos de trabajar con un fichero, puesto que su misión es «volcar» sobre el disco los datos que pudieran quedar en el *buffer*. De no hacerlo así, corremos el riesgo de perderlos, y de obtener información errónea cuando de nuevo trabajemos con ese mismo fichero.

Admite dos sintaxis: |CLOSE cierra todos los ficheros que estuvieran abiertos en el momento de ejecutar la orden. Por el contrario, |CLOSE,NF cierra solamente aquel cuyo número identificativo sea NF.

Los dos pequeños listados que acompañan al texto, se encargan, por una parte, de formatear el contenido de todos los registros de un fichero previamente creado por RANDOM-F, al valor que nosotros definamos (FOR-MAT).

Por otro lado, el programa VERIFY efectúa un vaciado del contenido de todos los registros del fichero selecçionado. Para utilizar uno u otro será preciso que el programa RANDOM.BIN esté presente en el disco de datos, o en la memoria central de nuestro ordenador. Para grabarlo basta con ejecutar la orden BASIC:

```
SAVE "RANDOM.BIN",B.&9C00,1792
```

FICHEROS ISAM (Indexed Secuential Acess Method)

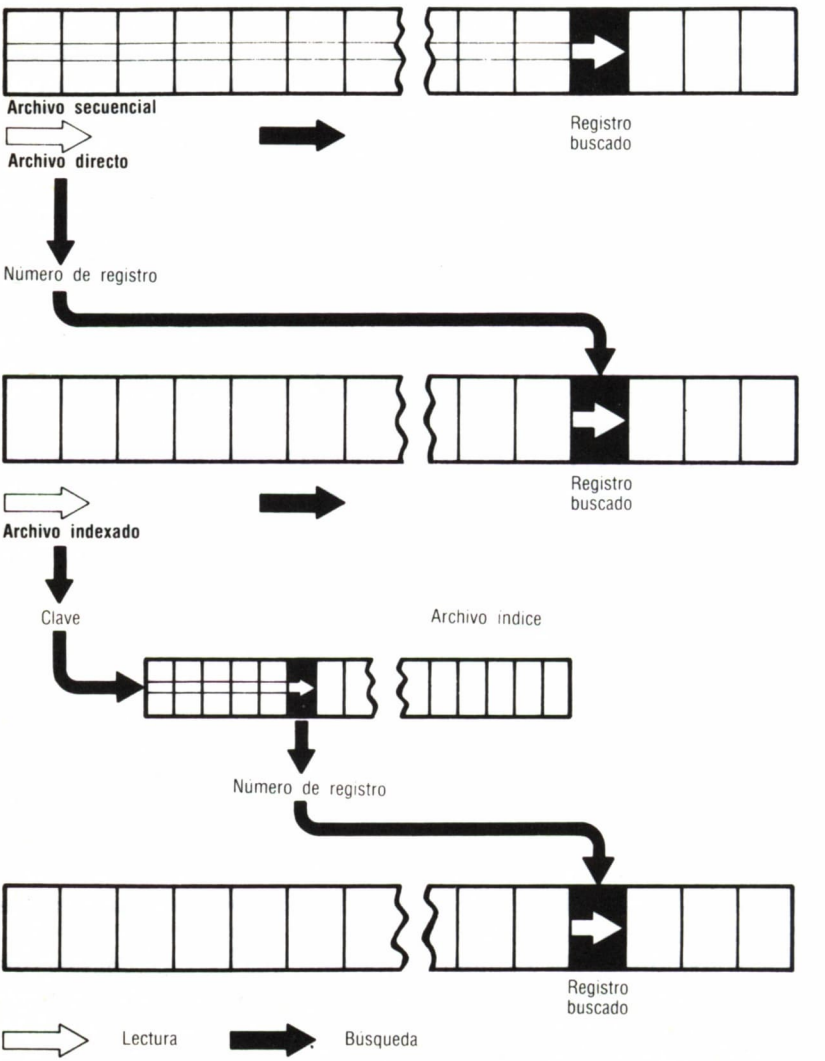
Este es el método de acceso secuencial indexado *(Indexed Secuential Access Method)*. El tipo de ficheros SIAM proviene de una mezcla entre el secuencial y el relativo.

Nos remitiremos al ejemplo de registros que estaban compuestos por 5 campos. Supongamos que contamos con 100 registros almacenados en este fichero. ¿Cómo haríamos si quisiéramos buscar a un cliente por su nombre?

Suponiendo que el fichero estuviese ordenado alfabéticamente por

el campo de datos NOMBRE, recorreríamos todos los registros hasta dar con el que coincidiese, por comparación con el campo de datos nombre que buscamos. Pensaremos que apenas hay diferencia con el almacenamiento secuencial de datos.

Hay algunas técnicas en las cuales nos es más fácil el proceso de búsqueda, por ejemplo, el árbol binario de búsqueda, pero se dan ac-



20.3.2. Sistemas de acceso a ficheros secuenciales, aleatorios e indexados.

cesos innecesarios a disco. Otro sistema es indicar un campo de datos del registro denominado KEY o INDEX; digamos que es como un «campo clave» que ha de estar siempre en memoria.

Conjuntamente con el contenido de este campo de datos, se encuentra el número de RECORD que le corresponda en memoria, accediendo directamente en el disco al registro que queremos.

Mediante esta medida ahorraremos memoria, debido a que no se conserva en la misma todo el fichero, es decir, 128 * 100 = 12.800 bytes, sino tan sólo las claves 45 * 100 = 4.500 bytes más los números de RECORD (100). Con este método ahorraremos bastante tiempo.

Por último, se pueden crear más ficheros de índice, aunque casi siempre es más que suficiente con uno.

MANEJO DE LA U. DE DISCO DESDE CÓDIGO MÁQUINA

ara programar en código máquina las operaciones que puede realizar el disco, existen dos procedimientos.

El primero de ellos consiste en programar todas y cada una de las operaciones que debe efectuar. Así, tendríamos que programar el encendido del motor, el posicionamiento de la cabeza, los tiempos de espera entre cada operación... etc. Ni qué decir tiene que esta forma de programación es cuando menos ardua y complicada.

Pero existe una segunda forma de programar en código máquina, bastante más sencilla que la anterior. Dado que las órdenes de disco son consideradas órdenes externas, existe para su manejo un sistema de vectores.

CONCEPTO DE VECTOR

La necesidad de utilizar vectores para el manejo del disco viene determinada por el hecho de que los ordenadores AMSTRAD no están en principio diseñados para utilizarlo. De esta manera existe el

grave inconveniente de que dependiendo del modelo de ordenador que poseamos, tendremos una versión distinta de sistema operativo.

Para solventar este problema se emplea el sistema de vectores. Mediante el uso de dicho sistema se asegura la compatibilidad, dado que la rutina es dirigida a través de una dirección en la RAM, que posibilita el salto a la rutina correspondiente. Entonces, siempre que la dirección para una rutina determinada sea la misma, y dicha rutina efectúe idéntica función, la compatibilidad queda segura. Así, es posible que funcionen programas con sistemas operativos que no son totalmente iguales.

Esto trae como ventaja adicional el poder dirigir el vector hacia rutinas propias, con lo que podremos modificar cualquier función que venga referida con vectores, siempre que ello sea preciso.

Rutinas Del DOS

Son trece las rutinas del DOS que mediante una ligera modificación se dirigen a la unidad de disco en vez de hacerlo al casete. Es decir, se modifica la dirección en que es llamada la rutina.

Mediante estos trece vectores, son llamadas desde el BASIC todas y cada una de las funciones que puede desarrollar el disco; como ya hemos dicho, estas rutinas pueden ser utilizadas en el código máquina.

Los nombres de dichas rutinas, según se denominan oficialmente en el manual de firmware del 464 son:

&BC77	CAS IN OPEN	Nos permite abrir un fichero para lectura.
&BC7A	CAS IN CLOSE	Cierra de forma ordenada un fichero abierto para lectura.
&BC7D	CAS IN ABANDON	Cierra de forma inmediata un fichero abierto para la lectura.
&BC80	CAS IN CHAR	Recoge un carácter del fichero abierto de entrada.
&BC83	CAS IN DIRECT	Lee en memoria un fichero de entrada de forma completa.
&BC86	CAS RETURN	Reescribe el último carácter.
&BC89	CAS TEST EOF	Comprueba si se ha alcanzado el final de fichero.

&BC8C	CAS OUT OPEN	Nos permite abrir un fichero para la escritura.
&BC8F	CAS OUT CLOSE	Cierra de forma ordenada un fichero abierto para lectura.
&BC92	CAS OUT ABANDON	Cierra de forma inmediata un fichero abierto para escritura.
&BC95	CAS OUT CHAR	Escribe un carácter en el fichero abierto de salida.
&BC98	CAS OUT DIREC	Escribe en un fichero una zona de memoria completa.
&BC9B	CAS OUT CATALOG	Es equivalente a la orden BASIC CAT.

CAS IN OPEN &BC77

Como es preceptivo, antes de poder leer datos del disco, es necesario el abrir un fichero de entrada. Para ello, utilizaremos esta rutina, no existiendo ninguna diferencia si el fichero del que queremos leer información contiene caracteres ASCII o un programa.

Lógicamente, para poder leer el fichero, es necesario determinar algunos parámetros.

El primero de ellos y quizás más importante es el nombre del fichero. En este caso, dado que el nombre de un fichero contiene ocho caracteres más tres de extensión, no puede transferirse por completo al registro HL. De esta manera, lo que se transfiere al registro HL es la dirección de la RAM en la que se encuentra el nombre del fichero que queremos abrir. El nombre del fichero lo podemos ubicar en cualquier dirección de la RAM, e incluso de la ROM, es decir en este caso, en las primeras 16 K de la memoria del CPC.

El segundo parámetro que debemos transferir es la longitud del nombre del fichero. Esta es una característica muy útil de la rutina, ya que no es necesario que el nombre y la extensión contengan respectivamente ocho y tres caracteres, puesto que la rutina alargará por sí misma el nombre, añadiendo espacios vacíos hasta adquirir la longitud precisa. Este parámetro debemos transferirlo al registro B.

En tercer lugar debemos transferir al registro DE la dirección de RAM a partir de la cual hemos reservado un buffer de 2.048 bytes. Este es el buffer en el que se escriben los datos a leer. En este caso, tampoco hay ninguna limitación en cuanto a la posición del buffer en la memoria.

Una vez hallamos transferido todos los parámetros a los registros, podremos efectuar la llamada a la rutina. El conjunto de instrucciones para abrir un fichero con el nombre «prueba.bas» tendría el siguiente aspecto:

```
        LD      HL,DIRNOM       Dirección del nombre
        LD      B,BUFF-DIRNOM   Longitud del nombre
        LD      DE,BUFF         Dirección del buffer
        CALL    &BC77           Llamada a la rutina
        RET
DIRNOM: DEFM    «prueba.bas»    Nombre del fichero
BUFF:   DEFS    &0800           Espacio para el buffer
```

Una vez ejecutadas las instrucciones anteriores, la rutina, ¿qué es lo que realiza y cuáles son los resultados?

Lo primero es comprobar si el nombre del fichero se ajusta a las reglas formales, como por ejemplo, la no inclusión de caracteres reservados. En caso de hallar alguno de ellos, el comando se interrumpe. Lo mismo ocurre en el caso de que el nombre sea mayor de 15 caracteres, incluyendo el número de USER, el nombre propiamente dicho, la extensión y los caracteres separadores.

Asimismo son convertidas a mayúsculas todas las minúsculas, y los nombres de fichero cortos son rellenados con espacios vacíos.

A continuación, se comprueba si existe en el disco un fichero con dicho nombre, y en caso de no encontrarse, se obtiene el mensaje de error —filename not found.

De todas maneras, en cualquier caso obtendremos algunos datos importantes en los registros tras haber efectuado la llamada. En caso de que el indicador de acarreo *(carry flag)* esté activado y el *zero flag* desactivado, será señal de que el CAS OUT OPEN ha llegado a buen término. En caso de encontrarse los dos indicadores desactivados, querrá decir que hemos intentado abrir un segundo fichero sin haber cerrado el anterior. Existe otra posibilidad: es cuando se encuentra activado el cero y desactivado el *carry flag*, lo cual indicará que el fichero no ha sido localizado.

Mediante el valor que tome el acumulador, sabremos el tipo de fichero hallado, en caso de que haya sido fructífera la apertura de fichero. De esta manera, si el valor del acumulador es 0, se tratará de un programa BASIC, pero si toma el valor &16, se tratará de un fichero ASCII.

En el par de registros HL, se encuentra tras la apertura del fichero, la dirección de la cabecera del mismo. En la cabecera se encuentra

información relativa al nombre, extensión, longitud y dirección en la que ha sido escrito el fichero. Sin embargo, en caso de ficheros ASCII, dicha cabecera es generada por el AMSDOS.

La dirección inicial del programa almacenado está disponible en el registro DE. Por su parte, la longitud del fichero se halla en el registro BC. Esta información carece de sentido en caso de ficheros ASCII.

CAS IN CLOSE &BC83

Mediante esta rutina se cierra un fichero abierto para lectura. Dado que sólo se puede abrir un único fichero para lectura, no es necesario el incluir ningún parámetro a esta rutina.

Después de llamarla, si el indicador de acarreo se encuentra activado será porque efectivamente se encontraba abierto un fichero para la lectura.

CAS IN ABANDON &BC7D

Aunque esta rutina puede considerarse idéntica a la anterior, su empleo está indicado para el cierre de fichero en caso de que exista algún error. Asimismo, es utilizada por el intérprete BASIC antes de ejecutar las órdenes LOAD, SAVE y CAT.

CAS IN CHAR &BC80

Esta rutina constituye uno de los dos métodos para poder leer datos de un fichero. Recoge sólo un carácter del fichero, no precisando parámetro alguno, dado que sólo puede encontrarse abierto un fichero para la lectura.

La rutina transfiere el carácter leído al acumulador. Mediante los indicadores de acarreo y cero podremos saber si la rutina ha llegado a feliz término, encontrándose en este caso activado el de acarreo y desactivado la *zero flag*. Si intentamos leer un carácter de un fichero que no haya sido abierto, los indicadores de cero y de acarreo se encontrarán desactivados. Asimismo, el acumulador tomará el valor &0E.

También estarán desactivados los indicadores cuando hayamos al-

canzado el final del fichero. Por su parte, el acumulador tomará el valor &1A, indicativo de END OF FILE.

Mediante esta rutina sólo podremos leer caracteres de modo secuencial. De esta forma, si queremos leer el carácter 50 y nos encontramos en el 70, tendremos que cerrar el fichero y abrirlo de nuevo.

CAS IN DIRECT &BC83

Este es el segundo método de leer de un fichero junto a la rutina anteriormente mencionada CAS IN CHAR.

Mediante CAS IN DIRECT podremos leer, y por consiguiente almacenar en la memoria, un fichero completo previamente abierto mediante la rutina CAS IN OPEN.

Esta rutina se emplea principalmente para cargar en memoria programas escritos tanto en BASIC como en Código Máquina.

Antes de efectuar la llamada a la rutina, debemos dotarla de un parámetro que indique la dirección para cargar el bloque de datos, que debemos transferir al registro HL.

```
        LD      HL,BUFFER       Dirección inicial de datos
        CALL    &BC83           Llamada a la rutina
        RET
BUFFER  EQU     $               Lectura dirigida aquí
```

Como de costumbre, obtendremos información sobre el éxito o fracaso de la rutina mediante los indicadores de cero y de acarreo.

Al igual que en las rutinas anteriores, si los datos fueron leídos de forma correcta, tendremos activado el indicador de acarreo, y desactivado el de cero.

Asimismo, los mensajes de error a que puede dar lugar esta rutina, son los mismos que en el caso de CAS IN CHAR.

Por su parte, el par de registros HL contiene la dirección ENTRY o dirección de cabecera de fichero. El significado de esta dirección será explicado más adelante en CAS OUT OPEN.

Una consideración que debemos tener siempre en cuenta, es la incompatibilidad entre las dos rutinas de lectura. En el caso de comenzar a leer datos de un fichero mediante DIC IN CHAR, no será posible continuar haciéndolo mediante CAS IN DIRECT. Lo mismo sucede en el caso contrario. Tampoco es conveniente leer dos veces el mismo fichero mediante CAS IN DIRECT, porque corremos el riesgo de que podamos perder datos en la memoria.

CAS RETURN &BC86

Con CAS IN CHAR un fichero solamente puede ser elaborado secuencialmente, existiendo la posibilidad de leer caracteres más de una vez.

Mediante el comando CAS RETURN, el último carácter leído se reescribe en el buffer y se encuentra disponible para la lectura. Una vez leídos los 20 primeros caracteres del fichero, tras la vigésima llamada a la rutina CAS RETURN, al efectuar el siguiente CAS IN CHAR se volvería a leer el primer carácter.

CAS TEST EOF &BC89

Mediante esta rutina podremos averiguar si ya hemos llegado al final de nuestro fichero.

Así pues, se lee el siguiente carácter mediante la llamada a la rutina CAS IN CHAR y compara con el valor &1A (26 decimal). Si por ejemplo, el valor del carácter leído no es 26, dicho carácter se reescribe en el buffer mediante la rutina CAS RETURN; para retornar de la rutina, lo haríamos pues, con los *flags* de carry activado y cero desactivado. Por consiguiente, si fuera hallado el valor 26, los dos *flags* se encontrarían desactivados.

Al igual que podemos saber que hemos llegado al final de nuestro fichero (como nos hemos referido anteriormente), esta rutina también la podremos utilizar para la lectura carácter a carácter de un determinado fichero. Esta rutina nos conducirá a mensajes de error al utilizarla con CAS IN DIRECT.

CAS OUT OPEN &BC8C

Todas las rutinas descritas hasta ahora, son válidas únicamente en el caso de que en el disco existan ficheros almacenados y queramos leer su contenido. Pero nada sabemos acerca de escribir datos (ficheros) en el disco. Sin embargo, las cosas no seguirán de esta manera mucho tiempo.

Las rutinas que explicaremos a continuación serán las que nos permitan escribir datos en el disco.

Lo primero que debemos hacer, es abrir un fichero para la escritura, cosa que podemos efectuar mediante el comando CAS OUT

OPEN. Al igual que sucedía en el caso de la lectura, debemos transferir diversos parámetros a dicha rutina.

En primer lugar, debemos transferir el nombre del fichero que deseamos abrir, al par de registros HL. Pero de igual forma que sucedía en CAS IN OPEN, lo que realmente transferimos a los registros HL es la dirección en la que se encuentra el nombre del fichero.

Asimismo, debemos transferir al registro B la longitud del nombre del fichero, a la vez que los registros DE reciben la dirección de un buffer de 2.048 bytes.

```
        LD      HL,DIRNOM           Dirección del nombre
        LD      B,BUFFER-DIRNOM     Longitud del nombre
        LD      DE,BUFFER           Dirección del buffer
        CALL    &BC8C               Llamada a la rutina
        RET
DIRNOM: DEFM    «prueba.ext»        Nombre del fichero
BUFFER: DEFS    600800              Espacio para el buffer
```

En este momento, la rutina comprueba que el nombre se ajuste a las reglas formales, y en su caso, lo completa mediante espacios vacíos, pasando las minúsculas a mayúsculas. A continuación, se sobreescribe la extensión del fichero con tres caracteres dólar ($$$), siendo entonces buscado el nombre en el disco. A partir del momento en que finaliza la rutina, disponemos de una cabecera de fichero, encontrándose la dirección de la misma en el par de registros HL. Ni qué decir tiene que todo esto será válido en el caso de no producirse errores.

Como de costumbre, sabremos por el estado de los indicadores si la rutina ha sido ejecutada de forma satisfactoria, encontrándose en este caso el indicador de acarreo activado, y por lo tanto, dispuesto un fichero para la escritura.

Tanto la entrada errónea del nombre, como la escritura fallida al comprobar el directorio, se notifican como error, no estando en este caso determinado el contenido de los registros HL.

Como ya hemos dicho anteriormente, si la ejecución de la rutina fue satisfactoria, encontraremos en los registros HL la dirección en la que se encuentra la cabecera del fichero. Esta cabecera tiene la misma configuración, tanto en la lectura como en la escritura. También son iguales las cabeceras del disco y del casete.

Los primeros 16 bytes de la cabecera contienen el nombre del fichero. Esto es consecuencia de la utilización del casete, ya que en el empleo del disco son necesarios 12 bytes.

Los bytes 16 y 17 no tienen ningún significado en el caso del disco.

En el uso de casete contiene el número actual de bloque y un indicativo para saber si se trata del último bloque del fichero.

Sin embargo, el byte 18 tiene bastante importancia, ya que indica el tipo de fichero. Está codificado en forma binaria, por lo cual cada bit tiene un significado concreto. En caso de encontrarse activado el bit 0, indicará que se trata de un fichero protegido, es decir, que ha sido grabado mediante SAVE«nombre.ext»,P.

Los bits 1, 2 y 3 son los que determinan el tipo de fichero. Será un fichero binario en caso de que se encuentren a cero los tres bits. Por contra, si están a uno los bits 1 y 2, se tratará de un fichero ASCII.

Por su parte, los bits 4 y 7 indican el tipo de versión, pero solamente los ficheros ASCII tienen el número de versión 1, es decir, el cuarto bit activado.

Los bytes 19 y 20 son solamente útiles en el caso de usar el casete, e indican el número de bytes en el bloque de datos actual.

Pero seguramente los bytes que más importancia tienen son el 21 y el 22. Estos contienen la dirección a partir de la que han sido escritos los datos. Este dato le es proporcionado a la cabecera después de haber escrito algo en el disco mediante CAS OUT DIRECT. Si el fichero almacenado es ASCII, los bytes tomarán el valor 0, ya que este dato no tiene ningún sentido en este tipo de ficheros.

El byte 23 toma siempre el valor &FF y solamente es útil en el uso del casete, indicando que el bloque de datos leído es el primero.

La longitud del fichero se encuentra almacenada en los bytes 24 y 25. Este valor es transferido el par de registros DE tras haber ejecutado CAS IN OPEN. De igual forma que en los bytes anteriores, en el caso de ficheros ASCII, este valor será siempre 0, ya que la longitud del fichero estará limitada por la capacidad de los discos.

Los bytes 25 y 26 son los últimos utilizados; contienen la dirección de inicio de los programas en código máquina. Cuando grabemos un programa en ese lenguaje debemos tener la precaución de poner la dirección de comienzo en estos bytes.

El resto de los bytes son puestos a cero por el sistema operativo, y no vuelven a ser utilizados; por lo tanto, quedan a nuestra disposición.

CAS OUT CLOSE &BC8F

Un fichero de salida ha de ser cerrado. Cada carácter pasa primeramente por el buffer de 2.048 bytes, creado con CAS OUT OPEN no llegando de un modo directo al disco. Si hubiera pues, un desbor-

damiento de dicho buffer, es decir, si el número de bytes superara 2.048, este buffer sería escrito en el disco. Así pues, si cerrásemos el disco encontraríamos 2.047 caracteres.

El resto existente en el buffer se escribe en el disco mediante la llamada a la rutina CLOSE, con lo cual el fichero se encontrará completo en el disco.

Si cambiásemos el disco con el fichero output abierto perderíamos 2.047 bytes.

Posteriormente, cambiaremos el nombre provisional por el original mediante la llamada a la rutina CAS OUT OPEN (con la extensión .$$$). Si hubiera con el mismo nombre otro fichero anterior, este último lo hallaríamos bajo la extensión .BAK.

Si alguna vez observamos en el directorio un fichero con la extensión (.$$$), se deberá a que el fichero no ha sido cerrado correctamente.

La rutina CLOSE no necesita parámetros de transferencia especiales dado que, también en la lectura, puede únicamente encontrarse un fichero abierto al tiempo.

Si la ejecución de CLOSE fue la correcta lo determinaremos mediante los *flags* de carry y cero, encontrándose pues, el *carry* activado y el cero desactivado.

CAS OUT ABANDON &BC92

El mensaje de error cuando el disco está lleno sería un claro ejemplo para definir esta rutina. En este caso, efectuaríamos un CAS OUT ABANDON, ya que los bloques dispuestos hasta ahora en el disco volverían a ser liberados.

En el directorio no aparecerá el nombre del fichero. De aquí deduciremos que deberían escribirse todos los datos otra vez en un disco con más espacio disponible.

CAS OUT CHAR &BC95

El AMSDOS dispone de dos sistemas distintos para escribir datos en el disco.

El primer sistema es mediante la rutina CAS OUT CHAR. Con esta rutina se escribe en el OPENOUT-BUFFER, situado en el acumulador. Si existiera un desbordamiento, los caracteres escritos en el

buffer hasta ese momento serían grabados en el disco, dando lugar a un nuevo espacio en el buffer.

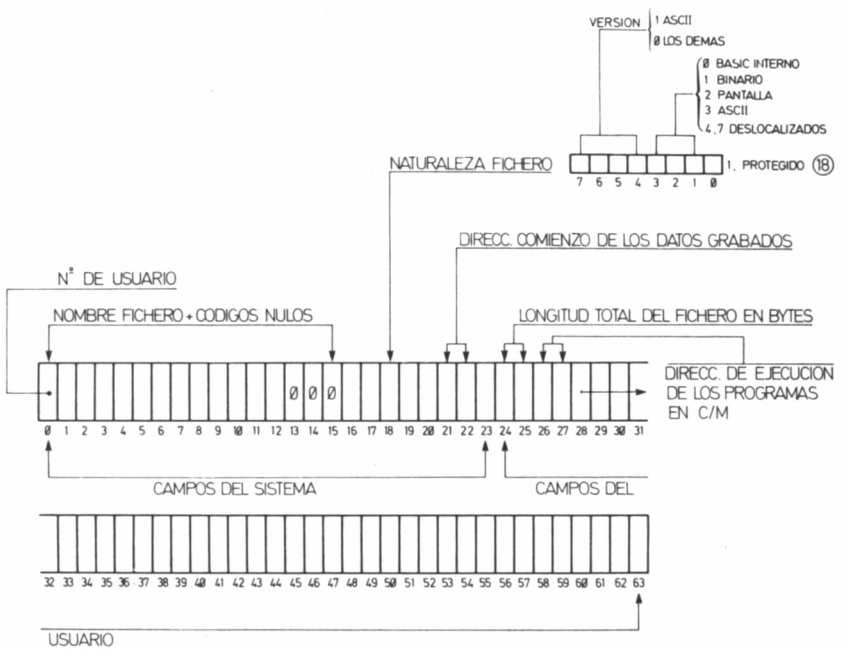
Casi todos los caracteres pueden ser escritos mediante esta rutina en un fichero. Pero hay que tener en cuenta que si luego el fichero es leído mediante la llamada a la rutina CAS IN CHAR, hay que tener cuidado al utilizar el carácter &1A (26 decimal), porque podía ser tomado en la lectura como criterio de EOF, aunque no sea el fin de fichero.

No ha de transferirse ningún parámetro a esta rutina como en las otras anteriores.

Después de efectuar CAS OUT CHAR, el *carry* está activado si el carácter fue correctamente escrito, pero si por el contrario éste se encontrara desactivado, significa que no abrimos el fichero de salida que es necesario.

CAS OUT DIRECT &BC98

Mediante esta rutina se da lugar al siguiente sistema de almacenar



20.4.1. Decodificación de la información de cabecera de un fichero.

datos en un fichero en disco. Esta rutina se emplea para el almacenamiento de zonas de memoria completas del CPC, por ello, se deberá transferir a esta rutina diversos parámetros.

El primer paso a seguir es el de dotar al par de registros HL con la dirección de inicio de la zona de memoria en la que vamos a escribir; a partir de ella serán transferidos los datos deseados en el disco.

El siguiente paso sería averiguar la longitud del fichero que deseamos. Cargaríamos pues, el par de registros DE con el número de bytes a almacenar. La longitud máxima de un fichero de este tipo es de 64k (65536 bytes).

Asimismo, es posible transferir la dirección de entrada (ENTRY) al par de registros BC. El contenido del par de registros anteriormente citados será escrito en los bytes 26 y 27 de la cabecera del fichero. Podemos también transferir un último parámetro al acumulador: el tipo de fichero. Este será escrito en la cabecera, al igual que el par de registros anteriores, pero en el byte 18.

No variaremos el sistema de escritura pasando de un método a otro; como por ejemplo, si escribiésemos un carácter con CAS OUT CHAR y utilizáramos posteriormente CAS OUT DIRECT. Aunque el mensaje de AMSDOS fuera O.K., tras el CAS OUT CLOSE aparecería el mensaje de error «file not open as expexted». Así pues, tras el fin de la rutina será cerrado el fichero.

CAS CATALOG &BC9B

Como ya sabemos, mediante la orden BASIC CAT, es posible obtener un índice de los ficheros almacenados en disco. La rutina encargada de hacerlo es CAS CATALOG. Para poderla utilizar en código máquina, será necesario transferir al par de registros DE la dirección inicial de un buffer de 2.048 bytes.

Aunque para poder ejecutar esta rutina no hubiera sido imprescindible dotarla del buffer anteriormente mencionado, se hizo necesario para la compatibilidad entre la versión del casete y disco; no obstante, esto aportó una ventaja adicional.

Cuando se ejecuta la orden CAS CATALOG, se leen del disco todos los nombres de los ficheros almacenados, y son dispuestos en el buffer junto la información relativa al número de bloques ocupados; aprovechando estas circunstancias, los nombres de los ficheros son ordenados alfabéticamente, en contra de lo que sucede con el comando DIR, que los coloca en el orden en que fueron escritos en el disco.

AMPLIACIÓN DE ORDENES DEL AMSDOS

En el capítulo anterior, dimos un repaso de las rutinas correspondientes a las órdenes que se usan indistintamente para el casete y para el disco.

En el presente capítulo, explicaremos las rutinas correspondientes a las órdenes de la ampliación, que son únicamente válidas en el caso de disponer de unidad de disco. Todas estas órdenes son las que en BASIC van precedidas de la barra vertical (|).

Por supuesto, estas rutinas son utilizables también en la programación en código máquina, aunque en este caso no existe la posibilidad de acceder a ellas mediante vectores.

ACCESO EN CÓDIGO MÁQUINA A LAS RUTINAS DE LA AMPLIACIÓN

Cuando queramos ejecutar cualquiera de las rutinas correspondientes a la ampliación de órdenes del AMSDOS, es necesario conocer la dirección en la que podemos hallar dicha rutina.

Dado que las rutinas pueden encontrarse tanto como residentes (RSX) en la RAM, como en las primeras 16 K correspondientes a la ROM del CPC, nos debe ser proporcionada la dirección en la que podemos encontrar la rutina. De ello se encarga una rutina especial del Kernal del CPC, para lo cual debemos darle el nombre exacto de la rutina que deseamos utilizar.

Para poder ejecutar la rutina especial de forma satisfactoria, debemos transferir al par de registros HL la dirección de la RAM en la que se encuentra el nombre de la rutina que deseamos utilizar. A continuación, haremos la llamada a la rutina KL FIND COMAND, cuyo vector se encuentra en la dirección &BCD4 de la RAM del CPC.

Una vez que hayamos ejecutado la rutina, nos serán devueltos los siguientes parámetros:

Si no fue hallada la rutina, el indicador de acarreo se encontrará desactivado. También puede ocurrir que la ampliación no se encuentre aún inicializada.

Esta última situación aparecerá en caso de encender en primer lugar la unidad central, para a continuación conectar la unidad de disco. Para solucionar esta solución bastará con efectuar un reset.

Pero, si por el contrario, el indicador de acarreo se encontrara activado tras ejecutar la rutina KL FIND COMAND, dispondremos de la dirección de la rutina en el par de registros HL. La dirección de ROM precisa se encontrará en el registro C.

El que se encuentren en esos registros las direcciones precisas para la llamada a la rutina no es producto de la casualidad. Existe otra rutina del Kernal, denominada KL FAR PCHL, que posibilita la llamada a cualquier dirección de la ROM o de la RAM como subprograma.

Para poder ejecutar correctamente la rutina KL FAR PCHL, la dirección de la rutina debe ir en el par de registros HL, mientras que la dirección de selección de ROM se hallará en C. Precisamente en estos registros es donde posiciona al rutina KL FIND COMAND las direcciones que necesitamos, de forma que hace mucho más sencilla la llamada.

Un pequeño programa en ensamblador para llamar al comando /DIR tendría el siguiente aspecto:

```
            LD      HL,COMANDO      Dirección del comando
            CALL    #001B           Llamada KL FIND COMMAND
            RET     NC              Comando no hallado
            XOR     A               Ningún parámetro
            CALL    #1B             Llamada a KL FAR PCHL
            RET
COMANDO:    DEFM    «P»,«M»+#80     Nombre del comando
```

En este ejemplo no ha sido necesario el transferir ningún parámetro, ya que la orden elegida no los necesita, debido a lo cual el acumulador se pone a cero (XOR A). Por otra parte, el nombre del comando debe tener a 1 el bit 7 de su última letra, lo cual equivale a sumarle 128 decimal (80 hexa.).

Gracias a las dos rutinas del Kernal que hemos utilizado, se ha simplificado mucho la programación. La cosa se complica algo más en caso de que sea necesaria la transferencia de parámetros para poder ejecutar la rutina. A continuación vemos un ejemplo en el que es necesaria la transferencia de un parámetro.

```
10              LD      HL,COMANDO      Dirección del comando
20              CALL    #BCD4           Llamada KL FIND COMMAND
30              RET     NC              Comando no hallado
40              LD      A,1             Transferir un parámetro
50              LD      IX,PARAM        Número de user buscado
60              CALL    #001B           Llamada a KL FAR PCHL
70              RET
80 COMANDO      DEFM    «USE»,«R»+&80   Nombre del comando
90 PARAM        DEFW    0002            USER 2, por ejemplo
```

Para este ejemplo hemos escogido el comando |USER,2, que solamente necesita que se le transfiera un parámetro: el número de USER.

El número de USER es transferido en el registro IX. Como deben ser transferidas palabras de 16 bits, el número de USER lo hemos definido en la última línea como un valor de dos bytes.

Cuando es necesario transferir cadenas, como en el caso de los comandos |ERA, |REN y |DRIVE, la programación se complica un poco más.

Dado que no es posible transferir directamente cadenas, será necesario transferir el puntero de la variable, que se obtiene con la función @ (arroba). El puntero de la variable no es más que un indicador del llamado «descriptor de cadena», que indica su nombre, longitud y ubicación absoluta en la memoria.

Tomaremos como ejemplo el comando |ERA, dado que sólo necesita una cadena por parámetro (por ejemplo, «nombre.ext»).

```
LD      HL,COMANDO      Dirección del comando
CALL    #BCD4           Llamada KL FIND COMMAND
RET     NC              Comando no hallado
LD      A,1             Transferir parámetro
LD      IX,PARAM        Descriptor de variable
```

```
        CALL  #001B          Llamada a KL FAR PCHL
        RET
COMANDO: DEFM «ER»,«A»+#80   Nombre del comando
PARAM:  DEFW  DESVAR         Dirección del descriptor
DESVAR: DEFB  10             Longitud de la cadena
        DEFW  DIRNOM         Dirección del nombre
DIRNOM  DEFM  «nombre.ext»   Nombre del fichero
```

En el próximo ejemplo necesitamos transferir dos cadenas, puesto que el comando |REN necesita el nombre del fichero al que vamos a cambiar el nombre, además del nuevo nombre. Cambiaremos el nombre del fichero «prueba.ext» por el de «nuevo.ext».

```
        LD    HL,COMANDO     Dirección del comando
        CALL  #BCD4          Llamada KL FIND COMMAND
        RET   NC             Comando no hallado
        LD    A,2            Transf. 2 parámetros
        LD    IX,PARAM       Descriptor de variable
        CALL  #001B          Llamada a KL FAR PCHL
        RET
COMANDO: DEFM «RE»,«N»,+#80  Nombre del comando
PARAM:  DEFW  DESANT         Dir. descrip. antiguo
        DEFW  DESNUE         Dir. descrip. nuevo
DESANT: DEFB  10             Longitud de la variable
        DEFW  NOMANT         Dir. nombre antiguo
NOMANT: DEFM  «prueba.ext»   Nombre antiguo
DESNUE: DEFB  9              Longitud de la variable
        DEFW  NOMNUE         Dir. nombre nuevo
NOMNUE: DEFM  «nuevo.ext»    Nombre nuevo
```

Aunque en el ejemplo se han dispuesto los datos uno a continuación del otro, en la aplicación concreta los podemos ubicar en cualquier dirección de la memoria. Asimismo, el orden tampoco tiene importancia alguna.

En todo caso, en el volumen 17 de esta misma colección (CODIGO MAQUINA AVANZADO) encontraremos una amplia descripción de la forma de acceder a los comandos residentes (los precedidos por la barra vertical), así como de la creación de rutinas propias que los utilicen e incluso se invoquen de este modo.

ORDENES OCULTAS DEL AMSDOS

Aunque con las descripciones de rutinas y comandos efectuadas

hasta el momento, es posible programar en código máquina todas las funciones que se pueden desarrollar desde el BASIC, existen aún 9 órdenes o rutinas a las que podemos acceder únicamente desde el código máquina.

Estas rutinas no se mencionan en ningún lugar del manual, pero por suerte para nosotros están ahí, en algún punto de la memoria de nuestro CPC.

La totalidad de los comandos de la ampliación, poseen un nombre con una longitud de un carácter, y dado que debe estar activado el séptimo bit, los nombres de los comandos son:

&81 Message ON/OFF
&82 Drive parameter
&83 Disk format parameter
&84 Read sector
&85 Write sector
&86 Format track
&87 Seek track
&88 Test drive
&89 Retry count

MESSAGE ON/OFF

Mediante esta orden podemos desconectar los mensajes de error correspondientes a las órdenes ocultas de la ampliación. En particular, esta orden desconecta el mensaje que aparece en la pantalla reclamando la entrada de C, I o R.

Transfiriendo al acumulador un valor distinto de cero antes de la llamada a la orden, conseguiremos la desconexión.

Esta orden no puede ser utilizada en la versión de BASIC 1.0 (CPC 464), ya que en este caso dependemos de los mensajes de error de la pantalla. Esto es así porque no podemos disponer del número de error transferido al acumulador en esta versión de BASIC.

Sin embargo, en el BASIC 1.1 correspondiente a los CPC 664 y 6128 podemos incorporar esta rutina y averiguar cuál es el error presente en el acumulador, mediante ON ERROR GOTO y la variable reservada del sistema DERR.

DRIVE PARAMETER &82

La unidad de disco, para su correcto funcionamiento, tiene defini-

dos una serie de parámetros, como los tiempos de espera entre cambios de pista, o hasta que el motor alcanza la velocidad precisa, etc.

Mediante esta rutina, podemos modificar todos los parámetros que afectan al funcionamiento de la unidad de disco. Así, podremos cambiar o determinar de nuevo los períodos para el HEAD LOAD TIME, HEAD UNLOAD TIME, tiempo remanente de rotación, tiempo de espera entre cambios de pista y por último, el tiempo de espera hasta que el motor alcance la velocidad nominal.

Al contrario de lo que sucede con las demás órdenes de la ampliación, no podremos utilizar las conocidas rutinas del KERNAL para acceder a esta orden.

La circunstancia que imposibilita el salto mediante KL FAR PCHL, es que la orden *Drive parameter* espera encontrar en el par de registros HL el comienzo de la tabla para los datos de la unidad de disco, mientras que el KERNAL necesita en el par de registros referidos anteriormente la dirección de la rutina a la que hay que saltar.

Afortunadamente no está todo perdido. Existen otros métodos para llamar a una orden de la ampliación. Veamos un ejemplo:

```
            LD      HL,COMANDO      Dirección del comando
            CALL    #BCD4           Llamada a KL FIND COMMAND
            RET     NC              Comando no hallado
            LD      (FARADR),HL     Dirección rutina
            LD      A,C             Selección de ROM al acum.
            LD      (FARADR+2),A    También almacenar
            LD      HL,TABPAR       Tabla parámetros disco
            RST     &18             Simulación de CALL
            DEFW    FARADR          Vector a dir. FARADR
            RET
COMANDO:    DEFB    &82             Nombre del comando
FARADR:     DEFS    3               Espacio dir. de 3 bytes
NUETAB:     DEFS    7               Trans. 7 bytes a &82
HDUNLD:     DEFS    1               HEAD UNLOAD TIME deseado
HEADLD:     DEFS    1               HEAD LOAD TIME tras FDC
```

En este listado hemos sustituido la orden del AMSDOS CALL por la orden RESTART. Mediante RST podemos ejecutar cualquier dirección de la RAM o de la ROM, al igual que ocurre con KL FAR PCHL. Sin embargo, existe la ventaja de la orden RST &18, consistente en que no es necesario registro alguno.

Por este motivo, cuando utilicemos esta orden, podremos transferir parámetros tanto al par de registros HL como al registro C, cosa del todo imposible si utilizamos las rutinas del KERNAL utilizadas hasta ahora.

Para un correcto funcionamiento de RST debemos transferir los tres bytes correspondientes a la dirección (los dos primeros son empleados para la dirección), y el byte de selección de ROM, a la dirección FAR en la memoria de nuestro CPC, en el orden indicado y de forma conjunta.

Con respecto a la tabla que contiene los nuevos parámetros que vamos a dar a la unidad de disco, tendremos en cuenta las siguientes consideraciones:

El primer elemento de la tabla, indica el tiempo de espera que es necesario después de la puesta en marcha del motor que arrastra el rotor del disco. La magnitud de dicha espera se expresa mediante un número de 16 bits. Normalmente este tiempo es del orden de un segundo. El valor asignado a este elemento de la tabla se decrementa en una unidad por el ticker-event, y dado que la frecuencia del ticker-event es de 50 por segundo, asignar una magnitud de 50 a este parámetro supone un tiempo de espera de un segundo.

El segundo elemento de la tabla indica el tiempo que debe permanecer el rotor en marcha después de haberse efectuado el último acceso al disco. En este caso, también se emplea el ticker-event, y el valor está definido en 16 bits. Normalmente, se utiliza un valor de 250 que corresponde a 2,5 segundos.

El tercer elemento de la tabla no debe ser alterado. Se trata de un byte que toma el valor de &AF.

El cuarto elemento define el tiempo de espera necesario tras un cambio de pista. Al igual que los dos primeros, también está definido por 16 bits. El tiempo estandarizado es de 12 ms. y depende de la unidad de disco empleada.

Asimismo, debemos definir en la tabla otros dos valores, correspondientes a otros tantos tiempos de espera. Se trata en este caso de los valores para *Head load time* y para *Head unload time*. Los valores normalmente asignados para estos tiempos son de 16 y 32 ms. respectivamente.

Esta orden es ejecutada automáticamente por AMSDOS después de la conexión o del reseteado del CPC, así como cuando se pone en marcha CP/M. Los valores de la tabla utilizada tras el reset se encuentran en la dirección &C5D4 de la ROM del CPC.

DISK FORMAT PARAMETER &83

Mediante esta instrucción podemos determinar el tipo de formato del disco que se halle introducido en la unidad.

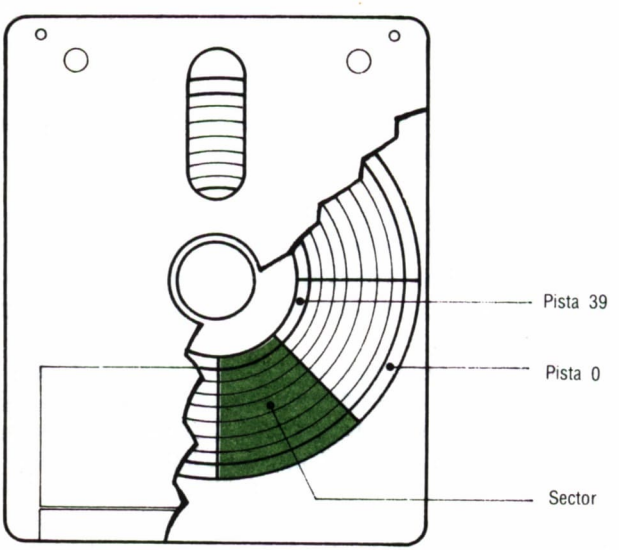
Como todos sabemos, los CPC pueden trabajar con tres tipos de formato de disco:

Formato de datos.
Formato de sistema.
Formato IBM.

En la ROM del AMSDOS se encuentra una tabla con los parámetros necesarios. Dependiendo del valor transferido al acumulador, la tabla precisa se transfiere a la RAM. Gracias a ello, podemos disponer de la tabla correcta cuando programamos en código máquina mediante acceso directo. En la programación de BASIC, se hace innecesario el uso de estas tablas, ya que el AMSDOS se encarga de ello.

La única diferencia existente entre los distintos formatos radica en los números de sector.

En el formato sistema o CP/M, existen 9 sectores en cada pista, conteniendo cada uno de ellos 512 bytes. Estos nueve sectores tienen los números de sector del &41 al &49, lo cual implica que el sexto bit se encuentra activado.



20.5.1. División de un disco en el proceso de formateado.

Para el formato de datos, se dan los números de sector del &C1 al &C9, estando en este caso fijados los bits sexto y séptimo.

Por último, en el formato IBM, los números de sector toman los valores del &01 al &08, por lo que sólo existen ocho sectores por pista.

De la anterior se desprende que si transferimos el valor 0, o cualquier otro que tenga los bits sexto y séptimo desactivados (cero lógico), encontraremos en la RAM la tabla de valores correspondiente para el formato IBM.

Consecuentemente, para conseguir el formato CP/M o sistema, debemos transferir un valor que tenga activado el bit sexto y desactivado el séptimo; tanto el valor &40 como el &73.

Ya para finalizar, obtendremos el formato de datos cuando se encuentren activos los bits sexto y séptimo, para lo cual utilizaremos indistintamente los valores &C0 o &FF.

READ SECTOR &84

Mediante esta orden podemos leer cualquier sector del disco.

Ello nos permite prescindir de las estructuras de datos conocidas, pudiendo programar ficheros relativos o ficheros ISAM. Esta orden es complementaria de la que explicaremos a continuación, que nos permite escribir en cualquier sector del disco.

Para poder utilizar de forma correcta esta rutina, debemos transferir algunos parámetros a los registros.

En primer lugar, indicaremos la unidad de disco seleccionada, transfiriendo al registro E un cero en caso de seleccionar la unidad A, y un uno en caso de que seleccionemos la unidad B.

A continuación, debemos indicar el sector en el cual deseamos que sea escrita la información. El número de sector debe ser transferido al registro C; además, el sector seleccionado debe existir realmente en el disco. Si queremos, por ejemplo, leer en el primer sector de una pista bajo formato CP/M, debemos transferir el valor &41.

Por supuesto, también debemos transferir el número de la pista. Esto lo haremos pasando el dato al registro D. Bajo CP/M, existen las pistas de la &00 a la &27.

Ya por último, debemos indicar en el par de registros HL la posición de memoria donde vamos a disponer los 512 bytes del sector.

Una vez que hallamos transferido todos los parámetros, los datos comenzarán a ser leídos y dispuestos en la zona de memoria especificada.

WRITE SECTOR &85

Esta orden puede considerarse como la inversa de la anteriormente descrita. Mediante ella, podemos leer información de cualquier sector de cualquiera de las pistas dispuestas en el disco.

Así podremos dar un vistazo al directorio o a las pistas reservadas para el CP/M en el disco.

Esta orden se programa de manera análoga a READ SECTOR, siendo necesario transferir los mismos parámetros a los mismos registros.

FORMAT TRACK &86

Mediante esta orden podemos formatear una única pista del disco.

Aunque su uso está bastante limitado, resulta muy útil para incluir rutinas de formateado sin tener que recurrir al programa correspondiente en CP/M.

Al igual que sucedía en las dos órdenes anteriores, FORMAT TRACK necesita que se transfieran diversos parámetros a los diferentes registros.

En primer lugar, debemos transferir el número de la pista que deseamos formatear. Este dato debe encontrarse en el registro D antes de la llamada a la rutina.

Otro dato necesario es la unidad de disco que vamos a utilizar, dato que debemos transferir al registro E.

En el registro C debemos indicar cuál es el primer sector a formatear.

Hasta aquí prácticamente no se diferencia nada esta orden de las dos anteriores. Sin embargo, el par de registros HL, se utiliza como puntero de una tabla, en la cual se encuentran cuatro bytes por cada sector a formatear, siendo los tres primeros los explicados hasta ahora (pista, unidad de disco y sector), más un cuarto que hace referencia a la magnitud del sector (bytes que contiene).

FORMATEO DE UNA PISTA

```
ORG   #5000        Dir. in. rutina format.
LD    E,DISCO      Seleccionar unidad disco
LD    D,PISTA      Número de pista
LD    C,#41        Número del primer sector
LD    HL,TABLA     Tabla 32 bytes para FDC
```

```
        RST   #18       Formato CALL &87
        DEFW FORMAT     Dir. de dir. 3 bytes FAR
        RET             Pista formateada
FORMAT: DEFW #C652      Dir. en AMSDOS de rut &87
        DEFB 7          Dir. de ROM select
TABLA:  EQU  $
SECT1:  DEFB PISTA      Núm. de pista sector ID
        DEFB CABEZA     Núm. de cabezal
        DEFB #41        Núm. del sector
        DEFB 2          Magnitud sector para ID
SECT2:  DEFB PISTA
        DEFB CABEZA
        DEFB #43
        DEFB 2
SECT3:  DEFB PISTA
        DEFB CABEZA
        DEFB #45
        DEFB 2
SECT4:  DEFB PISTA
        DEFB CABEZA
        DEFB #47
        DEFB 2
SECT5:  DEFB PISTA
        DEFB CABEZA
        DEFB #49
        DEFB 2
SECT6:  DEFB PISTA
        DEFB CABEZA
        DEFB #42
        DEFB 2
SECT7:  DEFB PISTA
        DEFB CABEZA
        DEFB #44
        DEFB 2
SECT8:  DEFB PISTA
        DEFB CABEZA
        DEFB #46
        DEFB 2
SECT9:  DEFB PISTA
        DEFB CABEZA
        DEFB #48
        DEFB 2
```

Esta orden permite el posicionamiento de la cabeza en una pista determinada. Hasta ahora, se indicaba a las rutinas qué pista debían leer, escribir o formatear, pero con esta rutina seremos capaces de posicionar la cabeza en la pista deseada.

Como es habitual es necesaria la transferencia de parámetros a la rutina, aunque en este caso sólo sean precisos dos.

En el registro E indicaremos la unidad de disco seleccionada y al D transferiremos el número de la pista en la que queremos posicionar el cabezal.

Mediante los indicadores de acarreo y de cero sabremos si la rutina llegó a feliz término, lo que será señalado con uno de los indicadores activados.

TEST DRIVE &88

Antes de leer o escribir información en el disco, es necesario saber si está disponible la unidad seleccionada. Mediante esta orden, podremos saber desde el código máquina si la unidad se encuentra realmente disponible.

Cuando el contenido del acumulador sea un cero, será señal de que la unidad A está efectivamente disponible, siendo este valor un uno cuando se trate de la unidad B. También tendremos información en los indicadores de acarreo y de cero, de si la rutina ha sido correctamente ejecutada.

RETRY COUNT &89

Una vez que tengamos el cabezal posicionado en la pista deseada, puede ocurrir que por alguna circunstancia la lectura sea infructuosa y haya que realizarla de nuevo. El motivo más frecuente es que la pista a la que hemos dirigido el cabezal no halla sido encontrada. En este caso, lo más lógico es realizar un nuevo intento antes de emitir un mensaje de error.

Mediante esta orden, podremos determinar el número de veces que se intentará encontrar la información antes de dar el temido DATA NOT FOUND.

Normalmente, el sistema operativo realiza diez intentos antes de darse por vencido.

Para modificar el número de veces que se intentará leer la información, debemos transferir el nuevo valor a través del acumulador, teniendo en cuenta que el cero se toma en este caso como el valor máximo, equivaliendo por tanto a 256.

Para hacer una prueba, podemos asignar mediante un POKE el valor 0 a la posición de memoria &BE66, que equivale a la transferencia de un cero al acumulador y efectuar la llamada a la orden &89. Si introducimos entonces un disco sin formatear en la unidad, oiremos durante bastante rato el chirriar de la unidad tratando de encontrar información.

Si a continuación asignamos un uno a la dirección &BE66, notaremos que la unidad se da por vencida mucho antes.

EL MICRO CONTROLADOR

Todas las funciones desarrolladas por el disco, serían poco menos que imposiles si no existiera dentro del ordenador un circuito específico para el control del mismo.

En los CPC, dicho circuito es el μPD765. Este circuito integrado por sí solo, constituye tanto el interface como el controlador de disco. Dado que es un circuito «muy inteligente», las labores de programación se ven considerablemente facilitadas para los diseñadores del sistema operativo.

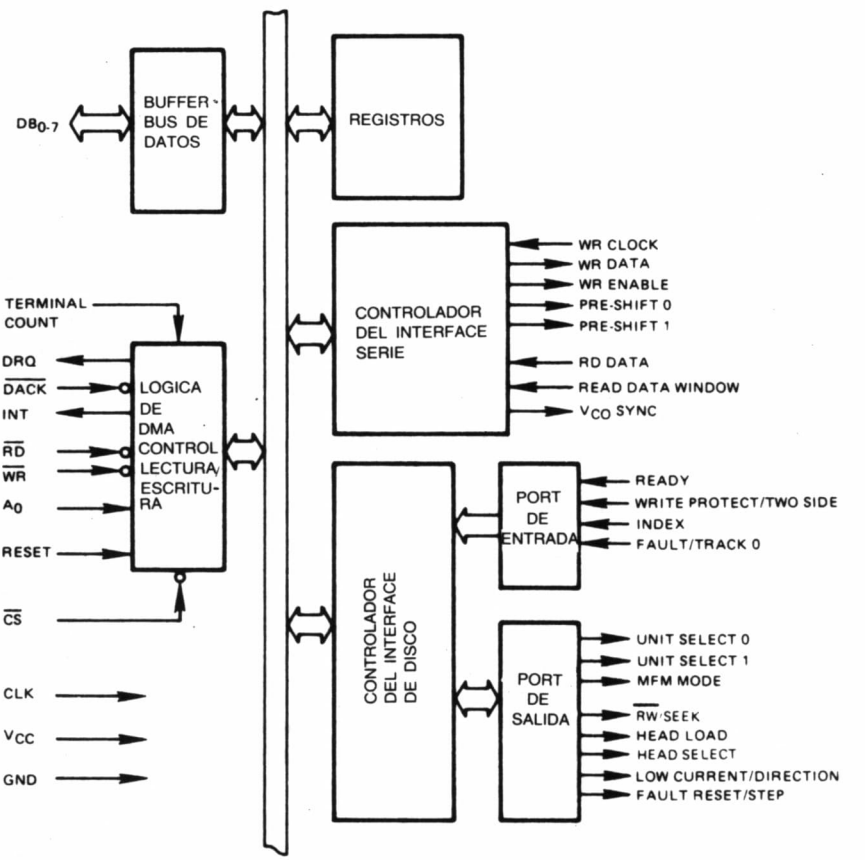
De igual forma, la circuitería necesaria para que este chip cumpla todas sus funciones es muy reducida. Estos dos factores redundan en un abaratamiento considerable del producto, sin disminuir en absoluto sus prestaciones.

A título de ejemplo, podemos citar la unidad de disco del COMMODORE 64. En ella, no se utiliza ningún circuito controlador específico, sino una CPU del tipo 6502. Dejando a un lado la irrisoria velocidad de transferencia, podemos constatar que la circuitería necesaria es mucho más voluminosa. Asimismo, la cantidad de software precisa para su control es ingente. El sistema operativo de COMMODORE para la unidad de disco es de 16K, contra los 8K que necesita la unidad del CPC.

EL DIAGRAMA DE BLOQUES

Un «diagrama de bloques» en electrónica, no es más que un esquema del circuito en el que los componentes se engloban en bloques funcionales.

El diagrama de bloques correspondiente al circuito controlador del disco es el de la figura. En él, podemos apreciar la ROM, la lógica de decodificación, el FDC (floppy disc controller), y, por último, la unidad de disco en sí.



20.6.1. Diagrama de bloques del controlador de disco.

La lógica de decodificción la componen una serie de circuitos lógicos, que deterinan la conexión o desconexión de la ROM y del FDC, dependiendo de la dirección existente en cada momento en el bus de direcciones, y del valor que tome el bus de datos.

La ROM del AMSDOS se conecta mediante un flip-flop cuando en la dirección de port &DFxx se da salida a un valor de &07. En este momento, la ROM del AMSDOS está activada en la zona de memoria que va desde &C000 a &FFFF para el acceso a lectura. Cualquier otro valor desconectaría la ROM del AMSDOS.

La ROM del controlador del disco tiene una capacidad de 16K. El sistema operativo del disco, como hemos dicho anteriormente, ocupa 8K. Las restantes 8K las emplea el lenguaje de programación LOGO.

En caso de que se pueda prescindir del LOGO, podemos sustituir la ROM por una EPROM (ROM programable), e incluir ella además del sistema operativo, una ampliación de órdenes que nosotros mismos programemos para nuestras necesidades específicas. De esta manera, el sistema operativo para el disco puede llegar a ser de hasta 16K.

La sustitución de la ROM por la EPROM no representa ninguna dificultad mecánica, ya que la ROM está insertada en un zócalo de 28 patillas, por lo cual sólo es necesario extraer la ROM e insertar la EPROM.

Un segundo flip-flop se encarga del control de los motores. Este flip-flop extrae los datos del bit de datos D0, mientras que el impulso de reloj (CLK) lo obtiene de los bits de direcciones A7, A8 y A10, y de las señales IORQ (petición de entrada/salida) y WR (escritura). Los bits de direcciones deben encontrarse a nivel lógico bajo, lo que produce que sea la dirección de port &FA7E la que dispare el flip-flop.

Asimismo, se utilizan idénticas señales para la decodificación del FDC. En este caso, el bit A8 debe estar a nivel lógico alto, lo que da como resultado que la dirección &BF7x sea la que dispare el flip-flop, y por tanto, genere la señal CS. El FDC tiene dos registros, y para diferenciarlos se utiliza el bit A0. Por lo tanto, se emplea la dirección &FB7E para el registro principal o de estado y la &FB7F para el registro de datos.

A través de la patilla 29 del FDC, se selecciona una de las dos unidades de disco. Si el valor de la señal del terminal 29 es nulo, la unidad activada será la B, mientras que de no ser así, la unidad seleccionada será la A.

CARACTERÍSTICAS DEL FDC 765

Dependiendo de la casa que fabrique o distribuya este circuito integrado, recibe distintos nombres. Así, para la casa japonesa NEC se llama μPD 765, para las norteamericanas ROCKWELL e INTEL se llama R 6765 y 8765 respectivamente.

El formato de datos que utiliza el FDC se corresponde con el 3740 de IBM en densidad simple, y con el sistema 34,también de IBM, para densidad doble. Por este motivo, no puede ser utilizado en los sistemas de COMMODORE o APPLE, entre otros.

Puede manejar los discos con medidas de 8, 5 1/4, y 3 pulgadas. Asimismo, tiene dos modos de funcionamiento.

El primer modo es el denominado DMA (acceso directo a memoria). En este sistema, el FDC toma el control de todas las funciones de transferencia de datos de la memoria al disco y viceversa.

El segundo método es el adoptado por el CPC. En él, las funciones de control de memoria las realiza la CPU central. Según este modo de operación caben dos variantes de funcionamiento.

En la primera de ellas se genera una interrupción por cada transferencia de datos, mientras que en la segunda (utilizada en el CPC), la CPU lee de forma regular la próxima acción a desarrollar de los registros del FDC. Esta segunda variante recibe el nombre de «polling» o escrutinio.

Las restantes características del FDC 765 son:

Longitud de sector programable
Datos de unidades programables
Conexión de hasta cuatro unidades de disco
Funcionamiento en DMA o en POLLING
Conectable a casi cualquier CPU
Alimentación única de 5 voltios
Reloj de una fase de 4 u 8 MHz.
Encapsulado de 40 patillas

PATILLAJE DEL FDC 765

Según la función desempeñada por los diversos terminales del circuito integrado, se pueden agrupar en cuatro conjuntos.

El primero de ellos, engloba a los terminales que hacen de interface con el ordenador.

El segundo conjunto lo forman aquellos terminales que guardan relación con el DMA.

Los terminales que sirven de conexión con los floppys constituyen el tercer grupo, y ya por último, el cuarto conjunto, lo constituyen los terminales de reloj junto con el de alimentación y masa.

TERMINALES DE INTERFACE CON EL ORDENADOR

RESET

Esta entrada se activa a nivel lógico alto. Por lo general, suele estar conectada a masa.

CS (chip select)

Esta es la patilla que selecciona el circuito y se activa a nivel lógico bajo. Antes de activar esta patilla debe estar definida la señal de lectura (RD) o la de escritura (WR).

RD (read, lectura)

Esta línea va conectada a la de igual nombre del ordenador. Se activa a nivel lógico bajo.

WR (write, escritura)

Cuando esta línea esté en nivel lógico bajo, el ordenador podrá escribir datos u órdenes en el FDC. Va conectada a la línea WR del ordenador.

AO (address line 0)

Mediante esta línea se distingue cuál de los dos registros del FDC se encuentra seleccionado. Generalmente, va conectada a la línea 0 (A0) del bus de direcciones, con lo cual se selecciona uno u otro registro mediante una dirección par o una impar. Cuando la entrada se encuentra en estado alto, queda seleccionado el registro de datos, y cuando está a nivel bajo se selecciona el registro de estado.

DB0-DB7 (databus 0-7)

Estas ocho líneas están conectadas a sus homólogas del bus de da-

tos del ordenador. Los datos pueden viajar en dos sentidos (entrada o salida). El sentido de los datos lo determina la CPU del ordenador o el control DMA mediante la activación de una de las líneas RD o WR.

INT (interrupción)

El FDC genera una interrupción por cada byte transferido. Esta conexión no se utiliza en el CPC.

Señales Para El DMA No Utilizadas Por Los CPC

DRQ (DMA request)

Mediante esta patilla, el FDC indica al DMA que debe conseguir un acceso a memoria, desconectándose la CPU del ordenador y tomando el DMA el control sobre el bus del sistema. Esta salida se activa a nivel alto.

DACK (DMA acknowledge)

Indica que comenzó la transferencia de datos, y que por tanto, el DMA se hizo cargo del bus. Esta salida se activa a nivel bajo.

TC (terminal count)

Poniendo a nivel alto esta entrada, se interrumpe la transferencia de datos en ambos sentidos. Generalmente, se emplea en modo DMA, pero también puede utilizarse en sistemas que hagan uso de las interrupciones. Se activa a nivel alto.

Patillas De Interface Con El Floppy

US0, US1 (selector de unidad)

Mediante estas patillas, se selecciona la unidad de disco. Gracias a un codificador de dos a cuatro líneas, pueden conectarse hasta cuatro unidades de disco.

HD (head select)

Mediante el FDC se pueden controlar unidades con dos cabezales, seleccionándose uno u otro mediante esta señal.

HDL (head load)

Esta señal es de uso casi exclusivo para unidades de 8 pulgadas, porque en estas unidades el motor de arrastre del disco suele estar siempre en funcionamiento, y con motivo de proteger el cabezal y el propio disco, se incorpora esta señal para levantar y bajar el cabezal.

IDX (índice)

Por esta patilla entra la señal recogida por el sensor óptico que indica el comienzo de una pista.

RDY (listo)

Esta señal la envía la unidad de disco al FDC para indicar que hay un disco en el interior girando a la velocidad nominal, y listo para ser leído o escrito.

WE (write enable)

Es una salida del FDC que indica, cuando está a nivel lógico alto, que pueden escribirse datos en el disco.

RW/SEEK (lectura escritura búsqueda)

Dado que existen determinadas señales que no son necesarias al mismo tiempo, el FDC tiene la peculiaridad de tener terminales con dos funciones. El número de estos terminales es cuatro, con un total de ocho señales. El FDC indica con qué juego de señales está trabajando en cada momento mediante el estado alto (seek, búsqueda) o el bajo (rw, lectura escritura).

FR/STP (fit reset/step)

Esta señal es la primera de las cuatro dobles de FDC. Puede emplearse tanto para borrar el error fijado en el flip-flop de la unidad de disco, en modo de lectura escritura, como en el modo de búsqueda, para enviar un pulso cada vez que el cabezal deba cambiar de pista.

LCT/DIR (corriente baja/dirección)

En modo de búsqueda, esta salida indica el sentido en que debe moverse la cabeza, ya que la señal anteriormente descrita sólo indica que la cabeza debe cambiar de pista, pero no especifica en qué dirección.

En modo de lectura escritura, sirve para delimitar la corriente del cabezal cuando se escribe en las pistas interiores del disco.

FLT/TR0 (fault/pista 0)

En modo de lectura escritura, esta patilla recibe la indicación de error procedente del flip-flop de la unidad de disco. En modo de búsqueda, recibe una señal producida por un sensor óptico o mecánico cuando el cabezal se encuentra en la pista 0.

CP/TS (write protect/cara dos)

Por este terminal entra la indicación de disco protegido para escritura, en modo de lectura escritura, y la indicación de disco de doble cara en modo de búsqueda.

WDA (escritura de datos)

Por este terminal son enviados en serie los datos hacia la unidad de disco, para ser escritos.

PS0, PS1 (precompensación)

Mediante estos dos terminales, se determina cómo deben ser escritos los datos en el disco en modo de doble densidad (MFM). Su existencia nace del hecho de que las pistas exteriores tienen más longitud lineal que las interiores. Se pueden definir tres modos de funcionamiento: *normal, early* y *late.*

RD (lectura de datos)

Por este terminal entran en serie los datos leídos en el disco.

RDW (read data window)

En esta patilla entra una señal procedente del PLL de la unidad de disco, y sirve como separador de datos.

VCO (vco sync)

Esta señal sirve para sincronizar el VCO de la unidad de discos.

MFM (modo MFM)

Mediante esta señal el FDC indica si está trabajando en densidad doble (nivel alto) o sencilla (nivel bajo).

TERMINALES PARA ALIMENTACIÓN Y RELOJ

Vcc (+5 voltios)

Esta patilla es la que recibe la tensión de alimentación, la cual debe ser de 5 voltios, con una tolerancia de ± 5 % sobre la nominal. El consumo máximo de corriente es de 150 mA.

GND (tierra)

Esta patilla debe estar conectada a masa (0 V).

CLK (reloj)

Por este terminal se le proporciona al FDC la frecuencia de reloj necesaria. Dicha frecuencia será de 4 MHz. para discos de 5 1/4 y 3 pulgadas, y de 8 MHz. para discos de 8 pulgadas.

WCK (reloj de escritura)

En este terminal se recibe la frecuencia de escritura de los datos, la cual deberá ser de 500 KHz. en densidad normal, y de 1 MHz. en densidad doble. El ancho del pulso debe ser de 250 ns.

LOS REGISTROS DEL FDC 765

omo ya apuntamos en el capítulo anterior, el FDC 765 posee dos tipos de registros, los registros de estado y el registro de datos.

El FDC dispone de un total de cinco registros de estado distintos. Al registro de estado principal se puede acceder en cualquier momento, poniendo la entrada A0 del FDC 765 a nivel lógico bajo. Este registro, al igual que los demás registros de estado, solamente puede ser leído. A los restantes registros de estado, únicamente se puede acceder en la fase de resultados, e incluso para acceder al último de ellos, es necesario un comando especial, como veremos más adelante.

Al registro de datos se accede poniendo la entrada A0 a nivel lógico alto. A través de este registro, el FDC recibe los parámetros necesarios para su programación. Este registro puede ser tanto leído como escrito.

PROGRAMACIÓN DIRECTA DEL FDC

El FDC 765 dispone de un repertorio de 15 comandos. Podemos diferenciar tres fases en la ejecución de un comando.

Primera fase. Programación.

En esta fase se transfieren todos los parámetros necesarios para que el comando pueda ser ejecutado. Dependiendo de qué comando se trate, será necesaria la transferencia de 0 hasta 9 bytes.

Segunda fase.

Una vez se hayan transferido la totalidad de los parámetros necesarios, se inicia la ejecución del comando.

Tercera fase.

En esta fase, el FDC informa acerca del resultado del comando, hasta con un máximo de siete bytes de respuesta distintos.

En algunas órdenes, el esquema anteriormente descrito puede no cumplirse, ya que existen órdenes que no necesitan informar del resultado, e incluso que no necesitan de la parte de ejecución.

Como complemento a la explicación de los distintos comandos, proporcionamos unas tablas que resumen los bytes de órdenes que es necesario transferir. Asimismo, se indica cuáles son los diferentes registros de estado que proporciona el FDC tras la ejecución del comando.

Las abreviaturas que utilizaremos en las tablas son las siguientes:

MT	Multitrack-bit: Si este bit se encuentra activado, la función multisector continúa en la segunda cara del disco. Sólo es útil en unidades de doble cara, por lo cual el AMSDOS siempre le da valor 0.
MF	MFM-mode-bit: Sirve para que el FDC trabaje en doble densidad. El AMSDOS siempre le da valor 1.
SK	Skip-bit: Sirve para saltar sectores borrados. No se emplea ni en CP/M ni en AMSDOS, tomando el valor 0.
HD	Head-select-bit: Selecciona la cara en unidades dobles. En el AMSDOS es siempre 0.
US 0,1	Unit select: Mediante estos dos bits seleccionamos la unidad. El AMSDOS asigna un 0 para la unidad A y un 1 para la unidad B.
R	Read: Lectura.
W	Write: Escritura.

LECTURA DE DATOS

Para poder efectuar una lectura de datos, es necesario transferir 9 bytes al FDC en la fase de órdenes. Una vez que se transfiera el último byte de órdenes, se activa la señal Head Load y se espera el tiempo que hayamos programado. A continuación, se leen los ID (identificadores de sector) hasta encontrar el identificador del sector buscado o hasta que el impulso de índice aparezca dos veces.

En caso de hallarse el identificador de sector buscado, se inicia la fase de ejecución. En esta fase, se leen los datos contenidos en el disco, y son enviados al procesador a través del bus de datos. Cada 26 microsegundos aproximadamente, hay un byte preparado para ser enviado al microprocesador.

Por el contrario, si se producen dos impulsos de comienzo de pista y no ha sido encontrado el sector en cuestión, se pasa directamente a la fase de resultados.

Una vez que se ha transferido el último byte del sector al microprocesador, es necesario mandar al FDC un impulso de Terminal Count (TC, pin 16), para que dé comienzo la fase de resultados, ya que sin este impulso, el FDC procedería a una lectura múltiple de sectores, leyendo la totalidad de los sectores del disco.

Una vez ejecutado el comando, el FDC proporciona 7 bytes en la fase de resultados. Estos deben ser leídos por el microprocesador en su totalidad, ya que hasta que no se lea el último, el FDC no aceptará ningún nuevo comando.

Los tres primeros bytes se corresponden con los registros de estado 0, 1 y 2. Estos registros contienen los datos más importantes sobre el éxito o fracaso de la lectura.

Los bytes siguientes, indican el número de pista, la dirección del cabezal, dirección de sector y longitud del sector.

ESCRITURA DE DATOS

En este comando también es preciso transferir 9 bytes en la fase de órdenes o programación.

Una vez que se han transferido todos los bytes de órdenes, comienza la ejecución del comando, de igual forma que sucedía en el comando anterior, con la única variación de que el FDC, una vez que encuentra el indicador de sector buscado, pide al procesador los datos que deben escribirse en el disco.

Asimismo, la fase de resultados es análoga a la del comando anterior.

Lectura De Datos Borrados

El FDC tiene la posibilidad de marcar como sectores borrados mediante el Data Address Mark, sectores que contienen datos. Estos serían ignorados en una lectura normal, pero mediante esta orden pueden ser leídos.

La lectura de sectores marcados como borrados, no difiere en nada de la lectura normal.

Escritura De Datos Borrados

Esta orden es la inversa de la anterior. Con ella, podemos escribir datos en un sector y marcarlo como sector borrado.

Exceptuando el Data Address Mark borrado, esta orden no difiere en absoluto con una escritura de datos normal. Puede ser muy útil para ocultar ficheros en el disco.

Lectura De Una Pista

Esta orden se corresponde con la lectura de un sector, con la salvedad de que lee todos los sectores de la pista.

Este comando queda finalizado cuando se lee el sector marcado como último, o cuando se reciben dos impulsos de comienzo de pista tras el inicio del comando.

Formateado De Una Pista

Se trata de una orden precisa de la transferencia de 6 bytes de órdenes. Una vez que el FDC haya recibido todos los bytes de órdenes, esperará el impulso que indica el comienzo de pista para proceder a formatear con todas las Address Mark, Gaps e IDs necesarias.

Además, el procesador debe proporcionar cuatro bytes más por sector que se formatea. Estos datos son el número de pista, número de cabeza, número de sector y número de bytes por sector. Normal-

mente, el número de sector se incrementa en una unidad cada vez que se formatea un sector, pero no existe ningún inconveniente en numerar los sectores según nuestra conveniencia, lo cual puede representar un ahorro de tiempo a la hora de acceder a un determinado sector.

LECTURA DEL ID

Mediante este comando, se puede leer del disco el siguiente ID posible. Durante su ejecución no hay comunicación entre el procesador y el FDC, excepto en la fase de resultados.

Este comando es especialmente interesante para saber en qué lugar del disco se encuentra la cabeza, puesto que en el ID está almacenado el número de sector y el de pista.

COMANDOS DE SCAN (COMPROBACIÓN)

Estos comandos tienen una función de verificación entre los datos escritos y a escribir.

Existen tres comandos de SCAN, dado que hay tres formas de comparación: «igualdad», «menor o igual» y «mayor o igual».

Cuando se hayan transferido los 9 bytes de órdenes, el FDC reclamará datos al ordenador, de forma simultánea a la lectura del disco, comparándolos del modo que le ha sido ordenado.

Este comando finaliza cuando se cumple la condición de prueba en todo el sector, cuando se haya verificado el último sector de la pista, o cuando el FDC reciba un impulso de TC en la patilla 16.

RECALIBRADO (BÚSQUEDA DE LA PISTA 0)

Mediante este comando se mueve la cabeza de lectura/escritura hasta que encuentra la pista 0. Se considera que la cabeza está en la pista 0 cuando el impulso de Track 0 así se lo indique al FDC, o bien cuando éste haya suministrado 77 impulsos de paso al motor del cabezal.

Dado que en este comando no existe fase de resultados, podemos saber el resultado del comando mediante el registro de estado principal.

SEEK (BÚSQUEDA DE UNA PISTA)

El FDC cuenta con cuatro registros de posición, uno por cada unidad posible, en el que almacena la posición del cabezal en cada momento. Estos registros son puestos a cero tras haber ejecutado el comando de recalibrado.

Al ejecutar el comando Seek, se compara la posición actual del cabezal de la unidad especificada con la posición final. En caso de que los valores sean iguales, no es necesario mover el cabezal. Si los valores no son iguales, el FDC pondrá la patilla 38 en estado lógico alto o bajo, según sea el resultado de la comparación, determinando de esta manera el sentido en que se moverá el motor.

Al mismo tiempo, por la patilla 37, saldrán un número de impulsos de paso de motor, que viene determinado por la diferencia absoluta entre el valor de posición actual y final.

LECTURA DEL ESTADO DE INTERRUPCIONES (REGISTRO DE ESTADO 0)

En funcionamiento no DMA, el FDC genera interrupciones en los siguientes casos:

Durante la fase de ejecución.
Al inicio de la fase de resultados.
Al finalizar un recalibrado o un Seek.
Si varía la señal Ready de alguna de las unidades.

Las dos primeras causas son fácilmente reconocibles por el procesador; las dos últimas son detectadas si se ejecuta el comando que nos ocupa. En este caso, la causa de la interrupción se deduce a partir del valor de STO.

Es prácticamente obligatorio ejecutar este comando después de un Seek, ya que el FDC no aceptará una nueva orden hasta no hacerlo.

LECTURA ESTADO DE UNIDAD (REGISTRO DE ESTADO 3)

Este comando es específico para poder acceder al registro de estado tres, al que no se puede llegar de otro modo.

El registro informa acerca del estado de la unidad elegida y podemos acceder a él mediante este comando en cualquier momento.

SPECIFY (ESPECIFICAR DATOS DE UNIDAD)

Mediante este comando, podemos programar todos los tiempos de espera que debe cumplir el FDC entre cada movimiento o acción del cabezal. Esta orden es la primera que debemos enviar al FDC, pues sin ella el controlador no sabe de qué modo debe trabajar.

En primer lugar, debemos definir el STEP RATE TIME (tiempo de espera entre dos impulsos de STEP). Este tiempo es distinto para cada tipo de unidad de disco, y gracias a esta orden podemos definir el tiempo conveniente.

En segundo lugar, debemos definir el HEAD LOAD TIME (tiempo de carga del cabezal). Este dato solamente es necesario para unidades de 8 pulgadas, ya que para unidades menores se carga el cabezal con la señal MOTOR ON.

Por último, debemos definir el HEAD UNLOAD TIME. Este es el tiempo que espera el FDC tras el acceso a disco hasta que la señal HEAD LOAD vuelva a ser desactivada. Este dato es prácticamente exclusivo de unidades de 8 pulgadas.

SIGNIFICADO DE LOS REGISTROS DE ESTADO

En el capítulo anterior, hacíamos una pequeña referencia de lo que eran los registros de estado, para poder hacer comprensible su programación. En este capítulo, haremos una descripción pormenorizada de cada uno de los registros de estado.

Como ya sabemos, solamente son accesibles directamente y en cualquier momento dos de los cuatro registros: el principal y el 3. Al primero de ellos se accede a través de la dirección que tiene reservada con A0 en estado lógico bajo, y al segundo de ellos mediante la orden específica para acceder a él.

A los otros tres registros, es decir, del 0 al 2, solamente podemos acceder en la fase de resultados.

REGISTRO PRINCIPAL DE ESTADO

En este registro se encuentran los datos más importantes referidos al FDC. A través del mismo se regula todo el protocolo necesario

entre el FDC y el procesador. El significado de cada uno de los bits de este registro es el siguiente.

Bits 0-3 DBBusy (unidad ocupada).

Estos cuatro bits están asignados a las cuatro unidades posibles. Se activan cuando se efectúa un SEEK o un recalibrado, siendo entonces imposible efectuar una lectura o escritura en la unidad ocupada. Sin embargo, es posible efectuar un SEEK o un recalibrado en cualquiera de las otras unidades.

Para desactivar estos bits, debemos enviar al FDC el comando «lectura de estado de interrupciones», borrando esta orden los bits si es que finalizó el comando.

Bit 4 CB FDC busy (FAC ocupado).

Este bit se activa cuando el FDC está efectuando una lectura o escritura, y no puede ocuparse en ejecutar otro comando. Permanece activado desde el momento que recibe el primer bit de órdenes hasta que se lee el último byte de la fase de resultados, siendo entonces desactivado.

Bit 5 EXM (modo de ejecución).

Este bit se encuentra activado si ha empezado la fase de ejecución. Sirve para diferenciar los datos de informe de sector de los bytes de la fase de resultados.

Bit 6 DIO Data input/output (entrada/salida de datos).

Mediante este bit se indica en qué sentido debe fluir la información. Si se encuentra activado, es porque el FDC tiene preparado un byte para el procesador, mientras que si se halla desactivado, el FDC espera que el procesador le proporcione un byte.

Bit 7 RQM Request For Master.

Este bit cuando está activado indica, que el FDC está preparado para la transferencia de un byte a través del registro de datos. Si se encuentra desactivado señala la imposibilidad de efectuar la transferencia.

El sentido en que se debe realizar la transferencia (entrada o salida) queda determinado, como hemos visto, por el bit 6.

REGISTRO DE ESTADO 0

Este es el registro de estado de interrupciones, dado que indica cuál fue la causa de la interrupción.

Bits 0,1 US *Unit select (unidad activa).*

Estos dos bits indican que la unidad se encuentra activa en el momento de la interrupción.

Bit 2 HD *Head address (dirección del cabezal).*

Mediante este bit podemos saber qué cabezal estaba seleccionado en el momento de la interrupción.

Bit 3 NR *Not ready (no preparado).*

Este bit se activa cuando la unidad elegida no está lista al efectuar una lectura o escritura. También se activa en caso de seleccionar el segundo cabezal si éste no existe.

Bit 4 EC *Equipment Check (error de equipo).*

Este bit indica, mediante su activación, si la unidad notifica un error. También estará activado en el caso de que la señal TRKO permanezca después de un recalibrado. Esto puede ocurrir en unidades que tengan 80 pistas, ya que en el recalibrado, el cabezal se mueve hacia la pista 0 solamente 77 pistas. Así pues, si el cabezal se encontraba en las pistas 78 al 80, se detectará un error y será necesario repetir el recalibrado.

Bit 5 SE *Seek End (fin de seek).*

Este bit se pone a uno tan pronto como finaliza el comando SEEK.

Bits 6 y 7 *Código de interrupción.*

En estos dos bits, el FDC informa acerca de cómo transcurrió el comando. Se pueden notificar cuatro circunstancias:

0 0 El comando finalizó de forma satisfactoria.
0 1 Comando interrumpido. En este caso, el comando fue iniciado, pero por alguna causa fue interrumpido. Puede su-

ceder que aparezcan errores de lectura del disco. En el CPC, aparece cada vez que se lee o se escribe un sector, dado que no se usa la señal TC y se tiene que programar el último sector de pista como último sector a leer. Por este motivo, la aparición de este mensaje no implica necesariamente que exista error.

1 0 Comando inválido. Indica que el comando fue invalidado por ilegal y ni siquiera pudo comenzarse. También se obtiene esta respuesta en caso de leer el estado de interrupciones en un momento en que no se ha producido ninguna.

1 1 Comando interrumpido. Este resultado se obtendrá si en el transcurso del comando la unidad deja de estar lista, como por ejemplo si retiramos el disco.

REGISTRO DE ESTADO 1

Este registro facilita información acerca del desarrollo de la fase de ejecución del comando.

Bit 0 MA *Missing Address Mark (pérdida del data address mark).*

Este bit se encontrará activado en el caso de que el controlador no haya encontrado el ID de sector cuando el disco haya efectuado una revolución completa. Asimismo, la falta el Data Address Mark, o bien encontrarlo borrado, se notifica de igual manera.

Bit 1 NW *Not writable (disco protegido).*

Si se inserta un disco protegido frente a la escritura, cuando ejecutamos los comandos de escribir sector, escribir sector borrado o formatear lista, el bit será activado.

Bit 2 ND *No Data.*

Cuando encontremos este bit puede ser por varias causas:

La primera de ellas es que el controlador no encuentre el sector indicado durante una escritura, lectura o scan.

Cuando exista un error en la suma de comprobación.

Y por último, si ejecutamos el comando «leer pista» y no se encuentra el sector de inicio.

Bit 3 *No utilizado. Siempre está a cero.*

Bit 4 OR *Over Run (tiempo de transferencia excedido).*

El procesador debe leer del FDC un nuevo dato cada 26 microsegundos. Si por la causa que fuera, el procesador no puede leer los datos a esa velocidad, el FDC tendría listo un dato antes de que el anterior fuera leído, y en tal caso se activaría este bit.

Bit 5 DE *Data Error (error en datos).*

Para detectar si los datos han sido almacenados correctamente, el FDC durante la fase de escritura genera de forma automática una suma de comprobación, que se almacena en el disco junto con los datos. Las sumas de comprobación se generan de igual forma en la lectura, y se comparan con las almacenadas en el disco. En caso de que las dos sumas no coincidan se activa este bit.

Bit 6 *No utilizado. Siempre está a cero.*

Bit 7 EN *End of Track (fin de pista).*

Cuando el FDC intenta acceder a un sector después del fin de pista programado, este bit queda activado.

Registro De Estado 2

En este registro de estado también se facilita información sobre el éxito o fracaso del comando, como ocurre en el registro de estado 1.

Bit 0 MD *Missing Address Mark in Data Field.*

Este bit queda activado cuando el controlador encuentra una Data Address Mark borrada o simplemente si no encuentra ninguna. Este bit se activa al mismo tiempo que el bit 0 del registro de estado 1.

Bit 1 BC *Bady Cylinder (pista errónea).*

Este bit se activa cuando el número de pista leído no coincide con el de la orden y es &FF.

Bit 2 SN *Scan not Satisfied (scan no satisfactorio).*

Si al ejecutar el comando Scan, no se encuentra ningún sector que correspondiese con las especificaciones, este bit quedaría activado.

Bit 3 SH Scan Equal Hit.

Cuando el resultado del comando Scan es de igualdad, este bit queda activado.

Bit 4 WC Wrong Cilinder (pista errónea).

Este bit lo activa el FDC cuando encuentra alguna diferencia entre el dato y el número de pista leído en el ID en la ejecución del comando de lectura. En el ID se encuentra el número de pista, ya que este dato se escribe en él durante el formateado.

Bit 5 DD Data Error in Data Field (error en campo de datos).

Este bit se activa ante errores de CRC, de igual forma que el bit 5 del registro de estado 1.

Bit 6 CM Control Mark.

Si durante la ejecución de un comando de lectura o de Scan, el FDC encuentra un Data Address Mark borrado, activa este bit.

Bit 7 No utilizado. Siempre se encuentra a cero.

REGISTRO DE ESTADO 3

Este registro como ya hemos comentado anteriormente, sólo puede ser leído haciendo uso del comando específico para ello. Contiene información acerca de la unidad elegida.

Bit 0,1 US Unit Select (unidad).

Estos bits tienen el mismo estado que las patillas 28 y 29 de selección de unidad (US).

Bit 2 HD Head Address (cabeza).

Al igual que ocurre con los bits anteriores, éste refleja en esta ocasión el estado en el que se encuentra la patilla 27 del FDC (head select).

Bit 3 TS *Two Side (una o dos caras).*

Este bit, por lo general, se encuentra en estado bajo en unidades sencillas, y en estado alto en unidades de doble cara.

Bit 4 T0 *Track 0 (pista cero).*

Este bit estará activado cuando el cabezal de lectura/escritura se encuentre en la pista cero.

Bit 5 RY *Ready (preparado).*

Este bit se pone a nivel alto cuando la unidad seleccionada envía la señal de Ready.

Bit 6 WP *Write Protected (protección contra escritura).*

Si este bit se encuentra activado, significa que el disco que se encuentra dentro de la unidad está protegido contra la escritura.

Bit 7 FT *Fault (fallo).*

Este bit indica cuando está a nivel alto que la unidad ha enviado la señal de Fault (fallo o error).

EL FDC DENTRO DE LOS CPC

Una vez visto el repertorio de comandos y funciones que puede desarrollar el FDC, nos damos fácilmente cuenta que no se aprovechan todas sus cualidades cuando trabaja en un CPC.

Solamente podemos conectar dos de las cuatro unidades que es capaz de soportar. También resulta imposible el funcionamiento con unidades de doble cara, y no se utiliza la señal HEAD LOAD. Pero aun con todo disponemos de una unidad de disco rápida, potente y económica.

En lo referente al control de la unidad de disco, el FDC recibe la inestimable ayuda de un circuito integrado específico. Se trata del separador de datos integrado SMC 9229 para el CPC 464, y del SMC 9216 en el CPC 664 y 6128.

El FDC responde a las direcciones de port &FB7E y &FB7F. La primera corresponde al registro principal de datos, y la segunda al registro de datos.

Por su parte, el *controller board* ocupa una tercera dirección. Si escribimos un 1 en la dirección &FA7E se pondrán en marcha los motores de las unidades conectadas, mientras que si escribimos un 0 se detendrán.

Dado que tampoco se utiliza la señal TC en las operaciones de escritura y scan, para evitar la lectura múltiple de sectores, se utiliza el siguiente truco: durante la programación del comando se igualan el número del último sector a leer con el número del sector a leer. De esta manera, el sector a leer resulta ser el primero y último, por lo que cuando es leído se da por finalizado el comando.

De lo anteriormente dicho se desprende que debemos ser cautelosos a la hora de emplear los comandos. En especial debemos ignorar el error indicado por el Bit 7 del ST1.

FORMATOS DEL DISCO

Como todos sabemos, existen tres tipos de formato de disco. Cada uno tiene unas características específicas pero todos ellos comparten algunas de ellas.

Las características comunes a los tres formatos de disco son:

40 pistas por disco.
Longitud de sector de 512 bytes.
64 ficheros cómo máximo.

Las características específicas del formato sistema son:

9 sectores por pista numerados del &41 al &49.
2 primeras pistas reservadas para CP/M.
Pista 0, sector &41: sector de arranque.
Pista 0, sector &42: sector de configuración.
Pista 0, sectores &43 a &47: no utilizados.
Pista 0, sectores &48 y &49 y pista 1 sectores &41 al &49: CCP y BDOS.

Las características específicas del formato data son:

9 sectores por pista numerados del &C1 al &C9.

Y por último, las del formato IBM:

8 sectores por pista.
1 pista reservada para el sistema.

ESTRUCTURA DEL DIRECTORIO DEL DISCO

El número máximo de ficheros por disco es de 64. La estructura de los datos en el disco viene determinada por el uso del CP/M.

Para acceder de forma rápida a cualquier fichero del disco, el sistema operativo genera automáticamente un directorio durante la fase de escritura, en el que se anota en qué pista y sector de disco ha sido escrita la información.

El sector es la unidad mínima de zona de memoria a la que podemos referirnos en el trabajo con discos. Sin embargo, existe una medida inferior, el RECORD, que contiene 128 bytes. Esto es así, porque en las primeras versiones de CP/M, se trabajaba con sectores de 128 bytes, y por motivos de compatibilidad, los sectores actuales son múltiplo de esta cifra. En nuestro caso, un sector contiene 4 RECORDS.

También existe un múltiplo de estas dos medidas; el Bloque. La utilización de bloques viene determinada por la necesidad de almacenar ficheros de gran extensión. De esa manera, sería enorme la cantidad de RECORDS que habría que anotar. Sin embargo, al utilizar Bloques, los records son asignados a cada bloque a razón de 8 por bloque, o dicho de otra manera, 2 sectores por bloque, y lo único que se anota es el número de bloque. Este es el motivo de ocupar 1K del disco cualquier fichero, aunque sólo mida realmente un byte.

Como ya se mencionó en capítulos anteriores, cada fichero abierto en el disco tiene una cabecera de 32 bytes. No es nuestra intención referirnos a ellos uno por uno de nuevo, por lo cual sólo mencionaremos aquellos que en este momento tienen interés.

El primer byte contiene el número de usuario (del 0 al 15); en algunos casos este byte toma el valor &E5. Significa que este fichero ha sido borrado, dado que el AMSDOS da de baja un fichero solamente en el directorio, aunque los datos siguen en su sitio hasta que son reemplazados por otros. Por tal moivo, en caso de haber borrado un fichero por error, podremos recuperar su contenido modificando mediante un monitor de disco el valor de usuario y dándole un valor permitido (del 0 al 15).

En los últimos 16 bytes (del 16 al 31) se anota la situación de los bloques siguientes, en el supuesto de un fichero que necesite más de un bloque. Esto a primera vista hace pensar que se puede almacenar un fichero con un máximo de 16 bloques (16K); en este caso es donde entran en juego los bytes 12 y 15. Si el byte 15 tiene un valor de 128, el AMSDOS supondrá automáticamente que todavía sigue una exten-

sión. En el byte 12 se numeran de forma creciente esas extensiones, concretando de esta manera el orden de lectura. De este modo, un fichero puede ser todo lo grande que sea preciso.

Por último, señalaremos que si el bit 7 del byte 8 de la entrada se activa, el fichero será de sólo lectura. Un fichero así marcado, no podrá ser borrado ni renombrado bajo AMSDOS o CP/M. Para activar este bit, podremos utilizar el comando CP/M STAT, o un monitor de disco, sumando 128 al valor que tenga el byte 8.

También se puede proteger de miradas indiscretas un fichero, activando el bit 7 del byte 9 de la entrada. La activación de dicho bit la haremos de igual forma que en el caso anterior. Es importante señalar que no debemos olvidar el nombre de los ficheros así protegidos, ya que no aparecerán en el directorio ni con CAT ni con DIR.

TABLAS RESUMEN
CONTROL DEL FDC

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
		LECTURA DE DATOS								
Comando	W	MT	MF	SK	0	0	1	1	0	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	---------- numero de pista ----------								Informacion ID antes
	W	---------- cabezal ----------								de ejecutar coamndo
	W	---------- sector ----------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador
Resultado	R	---------- estado 0 ----------								Contenido registros de
	R	---------- estado 1 ----------								estado
	R	---------- estado 2 ----------								
	R	-------- numero de pista ----------								Informacion de sector
	R	---------- cabezal ----------								tras ejecucion
	R	---------- sector ----------								
	R	------- longitud de sector ---------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
ESCRITURA DE DATOS										
Comando	W	MT	MF	0	0	0	1	0	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	---------- numero de pista ----------								Informacion ID antes de ejecutar coamndo
	W	---------- cabezal ----------								
	W	---------- sector ----------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDCy procesador
Resultado	R	---------- estado 0 ----------								Contenido registros de estado
	R	---------- estado 1 ----------								
	R	---------- estado 2 ----------								
	R	-------- numero de pista ----------								Informacion del sector tras la ejecucion
	R	---------- cabezal ----------								
	R	---------- sector ----------								
	R	---------- longitud de sector ----------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
LECTURA DE DATOS BORRADOS										
Comando	W	MT	MF	SK	0	1	1	0	0	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	---------- numero de pista ----------								Informacion ID antes de ejecutar coamndo
	W	---------- cabezal ----------								
	W	---------- sector ----------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador
Resultado	R	---------- estado 0 ----------								Contenido registros de estado
	R	---------- estado 1 ----------								
	R	---------- estado 2 ----------								
	R	-------- numero de pista ----------								Informacion de sector tras ejecucion
	R	---------- cabezal ----------								
	R	---------- sector ----------								
	R	------- longitud de sector ----------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
ESCRITURA DE DATOS BORRADOS										
Comando	W	MT	MF	0	0	1	0	0	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	--------- numero de pista ----------								Informacion ID antes
	W	------------ cabezal ----------------								de ejecutar coamndo
	W	------------ sector -----------------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador
Resultado	R	------------ estado 0 ----------------								Contenido registros de
	R	------------ estado 1 ----------------								estado
	R	------------ estado 2 ----------------								
	R	-------- numero de pista -----------								Informacion de sector
	R	------------ cabezal ----------------								tras ejecucion
	R	------------ sector -----------------								
	R	------- longitud de sector ----------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
LECTURA DE UNA PISTA										
Comando	W	0	MF	SK	0	0	0	1	0	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	--------- numero de pista ----------								Informacion ID antes
	W	------------ cabezal ----------------								de ejecutar coamndo
	W	------------ sector -----------------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador. El FDC lee todo lo que esta en la pista entre la perforacion de indice y la marca de EOT.
Resultado	R	------------ estado 0 ----------------								Contenido registros de
	R	------------ estado 1 ----------------								estado
	R	------------ estado 2 ----------------								
	R	-------- numero de pista -----------								Informacion de sector
	R	------------ cabezal ----------------								tras ejecucion
	R	------------ sector -----------------								
	R	-------- longitud de sector ----------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
FORMATEAR PISTA										
Comando	W	0	MF	0	0	1	1	0	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	--- longitud de sector ---								Bytes/sector
	W	--- sectores por pista ---								Sectores/pista
	W	--- espacio entre ID y datos ---								
	W	--- modelo de datos para sector ---								Byte de relleno
Ejecucion										FDC formatea la pista s completa.
Resultado	R	--- estado 0 ---								Contenido registros de
	R	--- estado 1 ---								estado
	R	--- estado 2 ---								
	R	--- numero de pista ---								En este comando esta
	R	--- cabezal ---								informacion no tiene
	R	--- sector ---								significado.
	R	--- longitud de sector ---								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
LECTURA DE ID										
Comando	W	0	MF	0	0	1	0	1	0	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
Ejecucion										Almacenar en el registro de datos la primera informacion correcta de ID de una pista.
Resultado	R	--- estado 0 ---								Contenido registros de
	R	--- estado 1 ---								estado
	R	--- estado 2 ---								
	R	--- numero de pista ---								Informacion sector ID
	R	--- cabezal ---								antes de la ejecucion
	R	--- sector ---								del comando.
	R	--- longitud de sector ---								

FASE	BUS DE DATOS R/W	D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
SCAN IGUAL										
Comando	W	MT	MF	SK	1	0	0	0	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	---------- numero de pista ----------								Informacion ID antes de ejecutar coamndo
	W	---------- cabezal ----------								
	W	---------- sector ----------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador.
Resultadc	R	---------- estado 0 ----------								Contenido registros de estado
	R	---------- estado 1 ----------								
	R	---------- estado 2 ----------								
	R	-------- numero de pista ----------								Informacion de sector tras ejecucion
	R	---------- cabezal ----------								
	R	---------- sector ----------								
	R	------- longitud de sector ---------								

FASE	BUS DE DATOS R/W	D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
SCAN MENOR O IGUAL										
Comando	W	MT	MF	SK	1	1	0	0	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	---------- numero de pista ----------								Informacion ID antes de ejecutar coamndo
	W	---------- cabezal ----------								
	W	---------- sector ----------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud-								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador.
Resultado	R	---------- estado 0 ----------								Contenido registros de estado
	R	---------- estado 1 ----------								
	R	---------- estado 2 ----------								
	R	-------- numero de pista ----------								Informacion de sector tras ejecucion
	R	---------- cabezal ----------								
	R	---------- sector ----------								
	R	------- longitud de sector ---------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
SCAN MAYOR O IGUAL										
Comando	W	MT	MF	SK	1	1	1	0	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	---------- numero de pista ----------								Informacion ID antes
	W	---------- cabezal ----------								de ejecutar coamndo
	W	---------- sector ----------								
	W	-------- longutud de sector --------								
	W	- ultimo numero de sector en pista -								
	W	- espacio entre datos ID y longitud -								
	W	long. de datos si long. de sector=0								
Ejecucion										Transferencia de datos entre FDC y procesador.
Resultado	R	---------- estado 0 ----------								Contenido registros de
	R	---------- estado 1 ----------								estado
	R	---------- estado 2 ----------								
	R	-------- numero de pista ----------								Informacipn de sector
	R	---------- cabezal ----------								tras ejecucion
	R	---------- sector ----------								
	R	------- longitud de sector ---------								

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
BUSCAR PISTA 0										
Comando	W	0	0	0	0	0	1	1	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
Ejecucion										Cabezal va a la pista 0

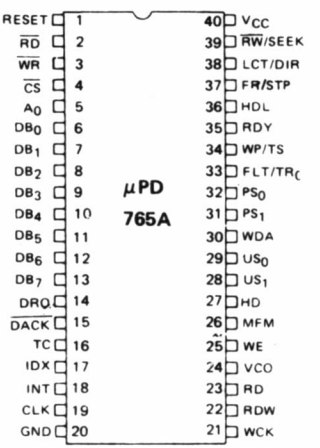
FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
BUSCAR PISTA										
Comando	W	0	0	0	0	1	1	1	1	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
	W	-------- numero de pista ----------								
Ejecucior										Cabezal va hacia la pista señalada

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
LEER ESTADO DE UNIDAD										
Comando	W	0	0	0	0	0	1	0	0	codigo comando
	W	X	X	X	X	X	HD	US1	US0	
Resultado	R	---------- estado 3 ----------								Informe estado FDC

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
DAR DATOS UNIDAD										
Comando	W	0	0	0	0	0	0	1	1	Codigo comando
	W	estep rate--><--elevar cabezal								Informe estado final
	W	cargar cabezal----------->no DMA								operacion busqueda por FDC

FASE	R/W	BUS DE DATOS D7	D6	D5	D4	D3	D2	D1	D0	OBSERVACIONES
LEER ESTADO DE INTERRUPCIONES										
Comando	W	0	0	0	0	0	0	0	0	codigo comando
Resultado	R	---------- estado 0 ----------								Informe estado al
	R	numero de pista tras orden busqueda								final operacion buqueda por FDC

CHIP CONTROLADOR DE VÍDEO



of Zilog, Inc.

Rev/3

459

Este volumen de la GRAN BIBLIOTECA AMSTRAD describirá con todo lujo de detalles las interioridades del periférico de almacenamiento masivo por excelencia: la unidad de disco. Comenzaremos ocupándonos de las funciones de carga, grabación, etc., conocidas por todos cuando las manejamos desde BASIC; a continuación, aprenderemos a controlar las extensiones del sistema de disco (RSX), analizando a fondo las nuevas órdenes, que mejor podríamos denominar «funciones ocultas» de AMSDOS. Para finalizar, una amplia descripción del chip controlador de disco, nos abrirá las puertas de un mundo de posibilidades ante esa auténtica enciclopedia en tres pulgadas que es un diskete.

GRAN BIBLIOTECA AMSTRAD

450 ptas.
(incluido IVA)
Precio en Canarias, Ceuta y Melilla: 435 ptas.