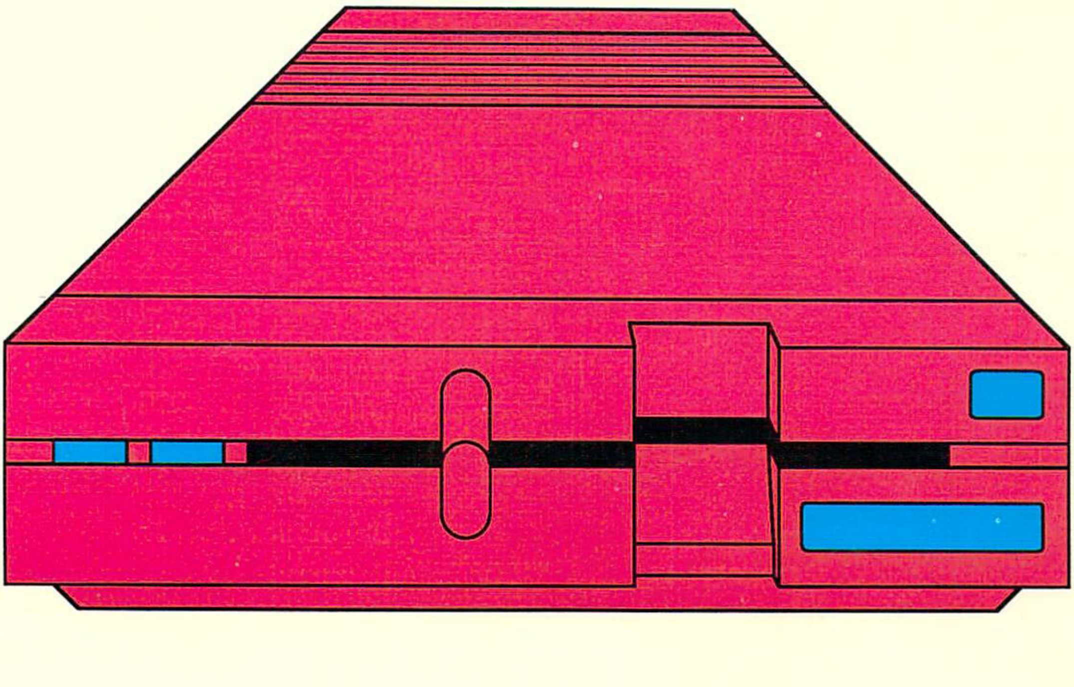
COMMODORE® 1571™

The essential reference for all 1571 users

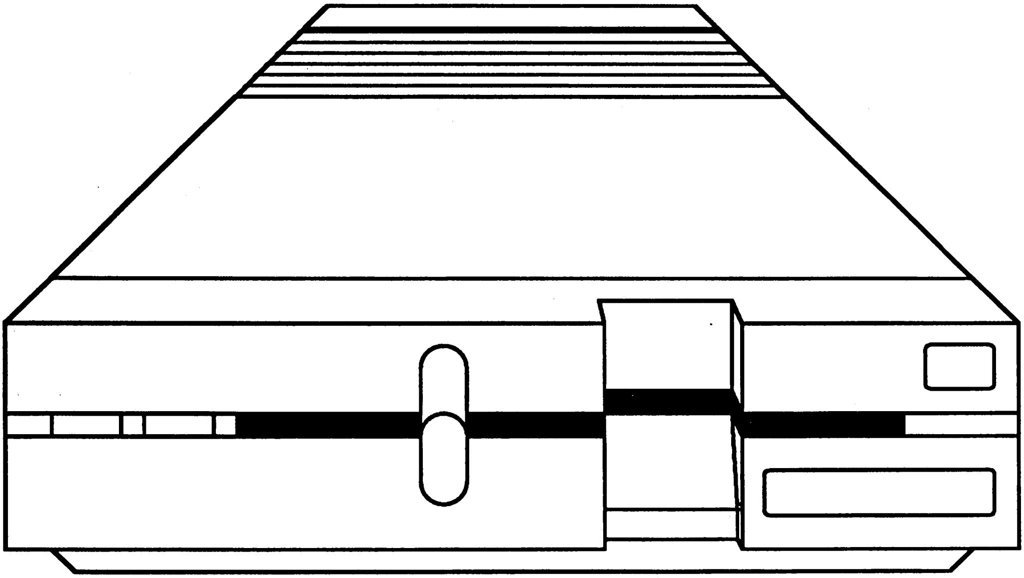
INTERNALS



A Data Becker book published by

You Can Count On Abacus Software

1571 INTERNALS



By Rainer Ellinger

A Data Becker Book

Published by

Abacus Software

Second Printing, June 1986
Printed in U.S.A.

Every effort has been made to insure complete and accurate information concerning the material presented in this book. However Abacus Software can neither guarantee nor be held legally responsible for any mistakes in printing or faulty instructions contained in this book. The authors will always appreciate receiving notice of subsequent mistakes.

ISBN 0-916439-33-X

Preface

Dear Reader,

With the 1570/1571 disk drive you have one of the most powerful 5 1/4" disk drives available for home computers. The 1570 is a single-sided disk drive that contains the electronics of the 1571, but is currently available only in Europe. The 1570/1571 processes two different Commodore disk formats and a number of different CP/M disk formats. In addition, the Commodore drives are probably the only drives which contain their own computers—they have independent microprocessor controllers.

This book is intended to help you get acquainted with all of the functions of the 1570 and the 1571. With this in mind, you will find a reader's guide following the table of contents. My goal is to lead you to the successful use of this disk drive— doesn't matter if you are a beginner or a professional. The 1571 Internals book is not only a tutorial guide, but above all it is also a reference work.

Expert programmers will find this book helpful. The ROM listing is in a class by itself. Never before has a ROM listing been so thoroughly documented. Two unique features of this listing are the entry points and calling address cross-references. You'll see these in Chapter 7.

Finally, I'd like to wish you the best when working with your 1570/1571 disk drive. Hopefully this book will offer you a much deeper understanding of the capabilities of the disk drive than can be obtained with the 1570/1571 user's guide alone.

Rainer Ellinger October, 1985

TABLE OF CONTENTS

Reader's Guide

1571 Internals is a very large book. A lot of information is packed in these pages. How exactly should you use the book?

The book has a table of contents, but this alone cannot make it a helpful handbook. For this reason, we've put together a Reader's Guide for this book. We've divided the audience into several categories, based on the reader's experience and previous knowledge. By reading the suggested sections, each reader will be able to gain the maximum benefits that this book has to offer. If you've:

1 Never worked with a computer before and are a complete beginner, read:

Sections 1.1, 1.2, 1.3, 1.4 and 1.5

2 Worked with other computers, have used the C-64 or C-128 without a disk drive and understand BASIC, read:

Sections 1.1, 1.3, 1.4, 1.5 and 5.1

3 Worked with other computers and disk drives, read:

Sections 1.2 and 1.3

4 Used the earlier 1541 disk drives, read:

Sections 2.1, 2.2, 2.3, 3.1, and 4

5 Worked with other computers and disk drives, and know machine language, read:

Sections 2.2, 3.1, 4, 5.2, 6, 7, and Appendices

6 Worked with 1541 and know machine language, read:

Chapters 6 and Appendices

All other sections should be used according to your areas of interest. Once you have the fundamentals, other information is available to the advanced user. The first chapters may also prove helpful to the professional for reference.

CHAPTER 1

FUNDAMENTALS FOR BEGINNERS

1.1 The first contact with the disk drive

1.1.1 After unpacking

Naturally you want to get started right away and begin using your disk drive. In spite of this, please be patient for a few moments as we cover this introductory section. First we will discuss setting up and connecting the drive. All our discussion are applicable to the 1570, a single sided disk drive with 1571 electronics (currently available in Europe) and the 1571 dual sided disk drive. In the following sections we will discuss connecting the disk drive and the data medium itself--the diskette. If you are already familiar with these things, you can move on to Section 1.1.3.

The following are included with the disk drive:

Power cord
Connector cable to the computer
Test/Demo disk
Instruction manual

First connect the 1570/1571 to the wall socket with the electrical cord. Be sure that the device is turned off. Next connect the drive to the computer using the black connector cable. One side of the connector cable plugs into the back of the computer as shown below:

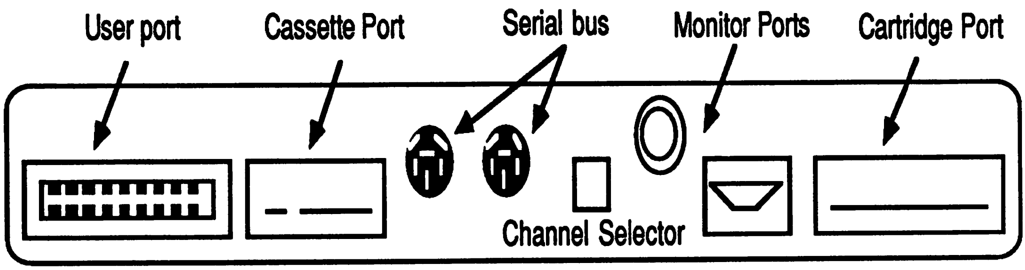


Figure 1 The back of the C-128

The other end of the connector plugs into one of the two jacks on the disk drive. Each device which you can connect to the computer (disk drive, printer, etc.) has two connectors. Otherwise you could operate only one peripheral from the computer because it had only one connector. One of the two connectors serves as an input and the other as output. A second disk drive or a printer would then be connected to this output connector on the 1570/1571. It does not matter which of the two connectors you connect to the computer. The important thing is that the other connector can be used only as an output. You cannot connect two computers to one disk drive.

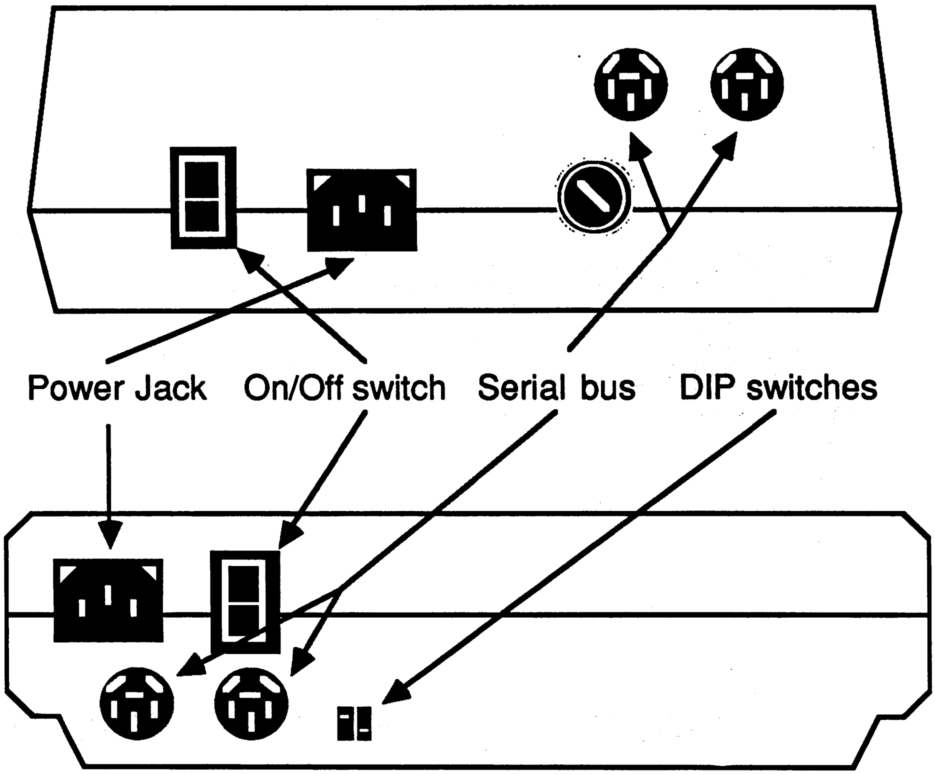


Figure 2 The back of the disk drive

If you're using a 1571, take a look at the two little switches, called DIP switches, on the back of the housing before you start using it. Their function is described in Section 1.2.1. Both of them should be up. On the 1570 these switches are inside the device and are already set correctly.

Now, when everything is ready, you can turn on the disk drive. On the 1570 a green LED lights up and on the 1571 a red LED lights up to indicate operation and the drive motor runs briefly. The green (1570) or red (1571) light indicates that the drive is turned on. If you observe the power-up process carefully, you will notice that the other LED lights up briefly. If all of this happens, then your C-1570/71 is functioning normally. If the red (1570) or green (1571) LED flashes, then the internal self-test routine has found an error.

The red (1570) or green (1571) LED also normally serves as a operating indicator. It indicates that the diskette inserted is currently being accessed. As long as this LED is lit you should not remove the diskette from the drive.

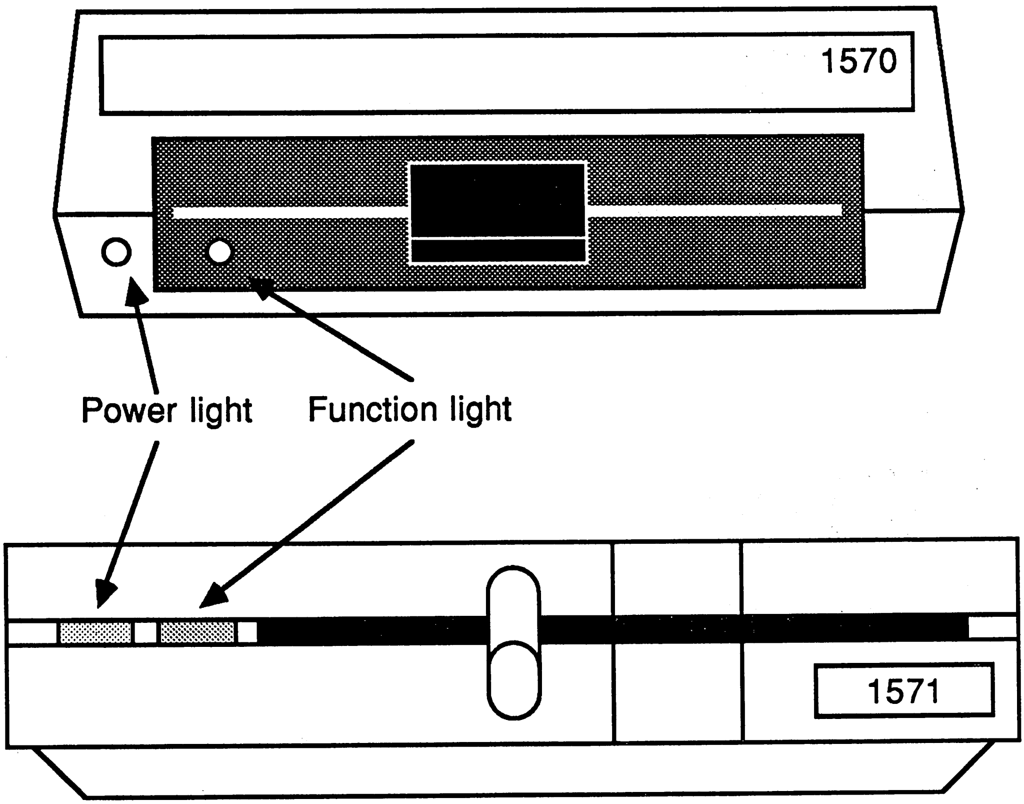


Figure 3 The front of the disk drive

1.1.2 What is a diskette?

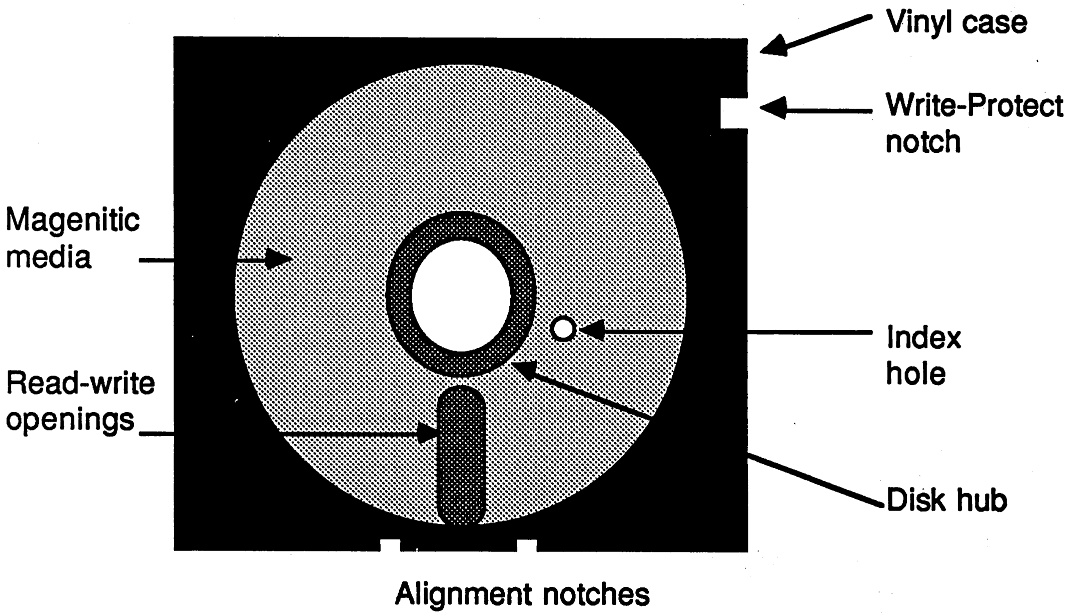


Figure 4 Diskette

Figure 4 shows a 5 1/4 inch diskette. The large opening in the lower section is immediately obvious. This is where the actual data media, a magnetic diskette, is visible. The read/write head in the drive, which transfers the data to and from the diskette makes contact with the media surface at this location.

A diskette is inserted into the drive with this opening going in first as in Figure 5. On the 1571, rotate the closing lever to the vertical position. On the 1570 press the closing latch downward. This seats the diskette properly over the drive hub as the motor runs for a few seconds to align the diskette better.

When the drive is in operation, the diskette rotates at about 300 RPM. The media is sealed in a plastic sleeve which protects the sensitive magnetic surface. The inside of the sleeve is lined with a cleaning cloth material that removes dust particles and other dirt. Keep in mind that the information stored on the diskette is only a few thousandths of an inch thick. Always handle the diskettes with care and never touch the actual media surface, only the protective sleeve. Your fingers contain oil and the cleaning cloth cannot remove it. Also remember to remove the diskette from the disk drive before you turn it off or on. Small uncontrolled voltages may damage important data.

The square notch in the right side is called the write-protect notch. As the name implies, it prevents accidental writing or erasing of data. By covering the notch with a write-protect tab (supplied with the diskette), the write mechanism on the drive is disabled.

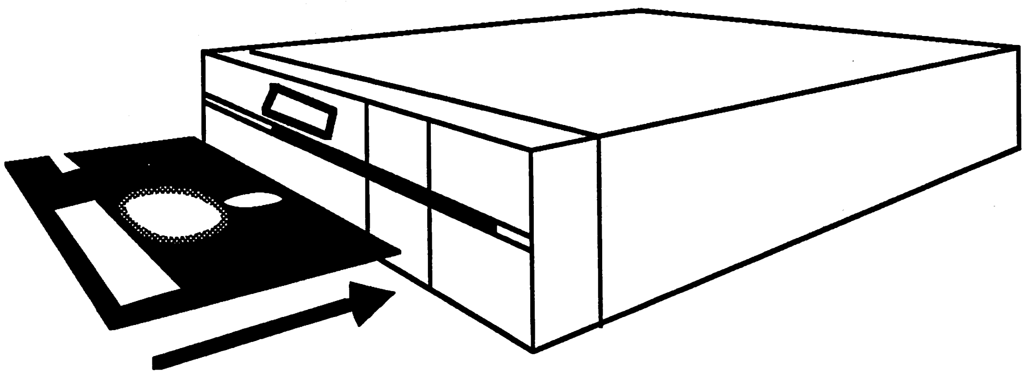


Figure 5 Correct position for disk insertion

Now we'll find out how the data is stored on the diskette. A diskette's surface is organized into *tracks*, as is shown in Figure 6. Tracks on a diskette are similar to the grooves on a phonograph record. The 1570/1571 drive can have a maximum of 40 tracks per side. Each track has a capacity of about 5000 characters.

Each track is organized into *sectors*. The number of sectors varies between 18 and 21 per track. Each sector has a capacity of 256 characters.

A special marker on the diskette is used to identify the sectors on a track. If you examine a diskette, you'll notice a small hole next to the hub. A photocell in the drive can sense when this hole is directly over the photocell. Here is where the first sector of the track begins. The position of the other sectors can be determined based on the rotation speed of the drive.

Does using the index hole have any advantages? Yes. It is flexible in that the size of each sector may be varied. By setting the start of the first sector, the position of other sectors may be determined regardless of their length. For the 1570/1571 the length is 256 characters.

The index hole method is used by the CP/M operating system. Diskettes which you use in C-64 or C-128 mode do not need the index hole. So that the drive still knows where a sector starts, special synchonization marks are written to the diskette magnetically. The drive recognizes these marks and thereby recognizes the start of a sector. But where is the first sector on the track? How are the sectors identified?

Each sector has a *header*. The header consists of information which precedes the actual data. In particular, the track number and sector number are found in the header. Using the header information, the drive electronics can "navigate" the diskette. To read a particular sector, the drive analyzes the next sector. It knows which track at which the head is currently positioned and can move the head to the desired track. Once there, the desired sector is found similarly.

Now, where do you write your data? Since there are more than 13300 sectors on a diskette, this could be an enormous task. But the 1570/1571 disk operating system (DOS) handles these details. The DOS keeps tabs on the sector usage, the file names and disk locations. We'll talk more about this later.

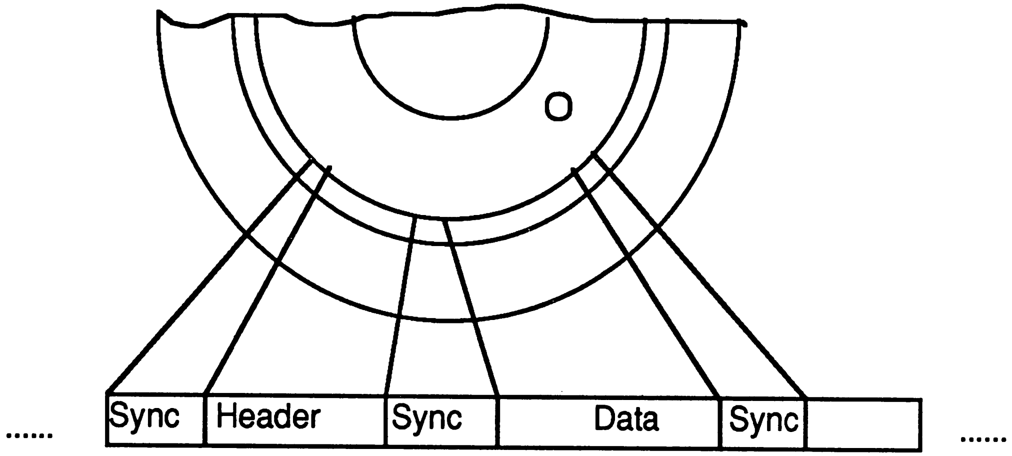


Figure 6 The diskette structure

1.1.3 Diskette formats

There are many ways to organize the storage of data on a diskette: index hole or sync mark; 128, 256, 512, or 1024 characters (bytes) per sector; varying the number of sectors per track; and others. The 1570/1571 writes 40 tracks per disk side. But there are also drives which can write 80 tracks per side (higher track density). Furthermore there are different recording processes. These are primarily different data packing factors and are therefore called single density and double density. There are also the tables about sector and disk allocation. Their organization depends on the type of computer used.

The result of these differences is that there are more diskette formats than there are computer manufacturers.

What type of diskette should be used for the 1570/1571? Any diskette that is rated for 40 tracks at double density and double sided can be used. This is often described on the diskette carton as:

2D (2sided, Double density)
or
DS/DD (Double sided, Double density).

New diskettes are always blank. Before using them to store data or programs, you must format the diskettes. More about this in Section 1.2.2.

Commodore format

Format	1541/1570	1571
Sides of diskette	1	2
Bits per sector, max.	307692	307692
Total number of sectors	683	1366
Number of free sectors	664	1328
Characters per sector	256	256
Number of sectors per track		
Tracks 1 - 17	21	21
Tracks 18 - 24	19	19
Tracks 25 - 30	18	18
Tracks 31 - 35	17	17

CP/M format

Sides of diskette	1	2
Bytes per sector	500000	500000
Number of sectors per track		
128 bytes per sector	26	26
256 bytes per sector	16	16
512 bytes per sector	9	9
1024 bytes per sector	5	5
Total number of sectors		
128 bytes per sector	1040	2080
256 bytes per sector	640	1280
512 bytes per sector	360	720
1024 bytes per sector	200	400

1.2 The disk drive and Commodore BASIC

1.2.1 From BASIC 2.0 to BASIC 7.0

To put the 1570/1571 to work, you must give it commands. This is not very complicated. Simply enter the command and press the <RETURN> key. This key tells the computer to execute the command. As you know, your C-128 accepts these commands in the BASIC language.

Just as there are various dialects of human languages, there are also different dialects of BASIC. Many of the fundamental commands are usually the same for all versions, but some commands differ in each version. In fact, the different versions of Commodore BASIC do not use the same commands for handling the disk drives. The table below lists the various Commodore computers and the versions of BASIC each contains. They are listed in order of appearance on the market.

Computer	Version
PET 2000	BASIC 1.0
CBM 3000	BASIC 3.0
CBM 8000	BASIC 4.0
VIC-20 / C-64	BASIC 2.0
C-16 / Plus 4	BASIC 3.5
C-128	BASIC 7.0

The version numbers is a measure of the power of the BASIC. BASIC 4.0 is somewhat more powerful that BASIC 2.0. But there are often exceptions to the rule since BASIC 3.5 should probably be renamed BASIC 4.5, because it is more powerful than the Commodore 8000's BASIC 4.0. As you can see, version 7.0 is ranked highest, exactly as far as the level of the C-128 BASIC—the most powerful BASIC that Commodore has produced.

For us, version 3.0 plays a deciding role. All versions of BASIC greater than 3.0 have easy-to-use disk commands. For the other versions, working with the disk drive is somewhat more complicated. The syntax of the versions which are less then 3.0 (referred to as BASIC < 3.0 hereafter) is also understood by the higher versions. The additional disk commands for BASIC > 3.0 (greater than BASIC 3.0) do not function on computers with lower BASIC versions. In the following sections, both forms of the commands are shown, that of BASIC < 3.0 as well as that of BASIC > 3.0.

Finally, there is a third option for using disk commands--in the built-in machine language monitor. The syntax of these commands is similar to syntax of BASIC < 3.0 and is also included.

So by sending a command to the 1570/1571, you can make it go to work. But what happens if two disk drives are connected to the computer? How does it know to which device the command applies?

Every device connected to the C-128 has a device number associated with it. Normally, the disk drive is assigned device number 8, a printer device number 4 and the cassette recorder device number 1. If you have a second disk drive connected, it cannot have the same device number 8. Instead, you must use a different device number.

On the 1571, there are two DIP switches on the back of the unit which determine the drive's device number. You can change the device number by changing the setting of the switches with a pencil point. On the 1570, the DIP switches are located inside of the drive housing. To change the device number you must unscrew the housing.

The following table lists the switch settings for changing the device numbers:

Switch 1 (left)	Switch 2 (right)	Device Number
up	up	8
down	up	9
up	down	10
down	down	11

To change the settings, you must turn the drive off, select the DIP switch settings and then turn the drive back on to effect a new device number.

Let's now take a look at data transfer between the computer and the disk drive. The 1570/1571 can not only store programs, it can also manage files. Let us assume that you are working with two files at once and you want to write new data in one of the files. When you send the data, how does the disk drive know what file it belongs to?

In order to solve this problem the 1570/1571 uses *data channels*. Each of these channels is used only for specific tasks. They are similar to radio channels. On one frequency there is police radio, on another the fire station and emergency, and so on.

On the 1570/1571 there are a total of 16 channels. Usually only three or four of these can be used at one time, however. The channels, like the individual devices, are assigned numbers. The following table shows the use of the channels:

Channel Number	Function
0	Load
1	Save
2 - 14	for Files
15	Command Channel

In order to activate a certain channel, you use the `OPEN` command on both the C-64 and C-128. The syntax of this command looks like this:

```
OPEN lfn,Y,Z,"data/name"
```

We haven't talked about parameter `lfn`. This is an arbitrary number between 0 and 255 and is call the *logical file number*. The logical file number is used by subsequent disk commands to refer to the opened channel. For example, to send data to the disk over the channel you would use a `PRINT#lfn` command where `lfn` is the logical file number from the `OPEN` command. The logical file number thereby shortens the specifications for other commands, making it easier to work with the disk.

These channel commands are especially important for file management. They will therefore be discussed in detail in Sections 1.4 and 1.5.

1.2.2 HEADER - Formatting a diskette

BASIC > 3.0:	HEADER "diskette name",Dx,Iyy,Uz
Abbreviation:	heA
BASIC < 3.0:	OPEN 1,z,15,"Nx:disk name,yy"
Monitor:	@z,N:x:disk name
Parameters (optional)	
Dx: x	= drive number
Iyy: yy	= 2 ID characters
Uz: z	= device address

In Section 1.1.3 we talked about *formatting*. Every new, blank diskette must be formatted before it can be used for data storage. Formatting places sync markers, headers, and sectors on the diskette.

If new diskettes are formatted, an ID must be specified. These two identification characters allow the disk operating system (DOS) to distinguish between diskettes and to determine if a new diskette has been inserted. This is why it is important to use a different character combination for each diskette. The ID information is placed in each sector header during formatting. In addition, the ID characters are also placed in the directory (title line) of the diskette. To change the ID later, the disk monitor described in Section 6.1 will be of help.

Not all character combinations are acceptable as an ID for BASIC 7.0. This is because the computer interprets the characters as a BASIC command and uses the corresponding abbreviation in place of the characters. But don't worry, there are quite enough combinations which are allowed. Together with the digits 0-9 there are 1296 possibilities. If there are 100 of these which you cannot use, it won't limit you too greatly. In addition, you can use the BASIC < 3.0 commands.

The following combinations of characters are not acceptable to BASIC 7.0: (Note upper and lower case)

```
on fn to aP aU bA bE bL bO bS bU cA cI cO dC dL dO dR dS dV eN fA fE fI
fR gR gS hE jO kE mO pA pE pL pO pU rC rD rE rR rS rU rW sC sL sO sP sS
sT sW tE tR vO wI xO aB aN aS aT cH cL cM cO dA dE dI eN eX fO fR gE gO
iN lE lI lO mI nE nO oP pE pO pR rE rI rN rU sA sG sI sP sQ sT sY tA tH
uS vA vE wA
```

If you use the header command without an ID, the diskette is not reformatted, but the data is erased. But just like complete formatting, all data will be lost in this process. This is why the computer asks "are you sure?" so that you must confirm the command before it is executed. If you answer the question with "y" for yes, it will perform the command. You can also use the header command in a program. In this case the user is not asked to confirm the command to format the diskette. The command is executed immediately, so you should program confirmation questions yourself in your own programs.

As you already know from Section1.1.3, the diskette formats in the C-64 and C-128 modes are not identical. CP/M diskettes have a completely different organization. The differences between the two modes result from the fact that the disk drive behaves like a 1541 drive when in the 64 mode or when connected to a C-64. If the computer is in the C-128 mode, the drive switches to the 1571 mode. The greatest difference between the two modes is the disk capacity. The 1571 uses both sides of the disk while the 1541 uses only one side, since it has only one read/write head. In spite of this, 1571 diskettes can also be used in the C-64 mode--provided a 1571 drive is used. The 1570 does not recognize a second side and always behaves like a 1541.

1.2.3 DLOAD/RUN - Loading and executing BASIC programs

BASIC > 3.0:	DLOAD "program name",Dx,Uy RUN "program name",Dx,Uy
Abbreviation:	dL / rU
BASIC <3.0: optional Parameter: x	LOAD "x:program name",y RUN "program name" is impossible
Monitor: aaaa = Starting addr. of program	L "x:program name",yy,aaaa auto-start is impossible
Parameters (optional):	
Dx: x =	Drive (0/1)
Uy: y =	Device # (4-15)

Now we're getting serious. These are the first commands for working with the diskette. Initially you will probably use your disk drive to store mainly programs.

Therefore we want to first discuss the commands with which you can read a program from the diskette into the computer. Its simplest form is:

```
DLOAD "program name"
```

The D in DLOAD stands for "disk." DLOAD is just a special version of the familiar LOAD command in which you do not need to specify the device address.

If the desired program is on diskette, it is loaded into the memory of the computer. If the program is not found, the computer responds:

```
FILE NOT FOUND
```

This also happens if the `DLOAD` command is used in a program. In addition, the program is interrupted and the computer returns to the direct mode.

You can try this command with the Test/demo diskette. Try to load the various programs from the diskette. To execute the program immediately after it is loaded, use `RUN` in place of `DLOAD`.

The keys <SHIFT + RUN/STOP> offer a still greater ease of use. If you press them together, the commands `DLOAD ":*"` (in abbreviated form) automatically appears on the screen and after it `"RUN"`. This causes the computer to read the first program on the diskette and run it.

Naturally there are differences between the load commands in the 64 and 128 modes. The most serious contrast is the transfer speed. In the C-64 mode the characters are transferred over the bus at a rate of 400 characters per second, while they travel at a rate of 3500 bytes per second over the C-128 bus. In practice this means that a graphic picture is no longer loaded in 20 seconds, but in 3.

In addition, the load command behaves differently when overlaying programs. While the C-64 normally "forgets" all of its variables, they all retain their values on the C-128. So you can easily divide large programs into several sections without problem.

1.2.4 DSAVE - Saving BASIC programs

BASIC > 3.0:	DSAVE "program name",Dx,Uy
Abbreviation:	dS
BASIC <3.0: optional Parameter:	SAVE "x:program name",y x
Monitor: aaaa/bbbb = Start & End addr. of program	S"x:program name",yy,aaaa,bbbb+1
Parameters (optional): Dx: x = Uy: y =	 Drive (0/1) Device # (4-15)

If a program is to be transferred from the computer memory to the diskette, we do it like `DLOAD` only here with the command `DSAVE`. If, for example, we want to save a BASIC program, we must find a suitable name for it. Let's assume it should be called `minitest`. The save command reads:

```
DSAVE "minitest"
```

The name may not be more than 16 characters long and a program with the same name may not exist on the diskette. In addition, there are some characters which may not be used in program and file names. These characters are control characters. If they are used, the program cannot be loaded because the drive will interpret the name as a command. Here are the prohibited characters:

```
, : ? * # & @
```

If you have changed your program and would like to save the new version with the same name, you can precede the name with the `@` character. For example:

```
DSAVE "@minitest"
```

The special function saves the new version and then erases the old version. Therefore there must always be enough free space on the diskette to hold a copy of the new version. Unfortunately, there are problems with this replace function. If the diskette is almost full, the function will not work correctly and your program may be lost. You should therefore use the @ with care--or better yet, not at all. In BASIC < 3.0 as well as in the monitor, a colon must follow the @ in order to separate it from the program name (such as `"@:minitest"`).

The save times on the 1570/1571 are not as fast as the loading time. Saving a program is no faster than on the 1541. In addition, saving is generally slower than loading because after each write the data must be checked to see if it was stored correctly on the diskette.

Loading and Saving Times

	Read	Write
C-64 10K Byte-program	0:27	0:30
C-64 10K Byte-file	2:25	2:45
C-128 10K Byte-program	0:03	0:25
C-128 10K Byte-file	3:05	2:50

1.2.5 DVERIFY - Verifying programs

BASIC > 3.0:	DVERIFY "program name",Dx,Uy,z
Abbreviation:	dV
BASIC <3.0: optional Parameter: x	VERIFY "x:program name",y,z
Monitor: aaaa = Starting addr. of program	V"x:program name",yy,aaaa
Parameters (optional):	
Dx: x =	Drive (0/1)
Uy: y =	Device # (4-15)
z : z =	0: relative load 1: absolute load

This command verifies that a program on the diskette is the same as the one in the computer's memory. It compares it with the one found in the memory of the computer. If they match, the computer responds `Ok`. If not, the message `Verify Error` will be displayed.

Historically `DVERIFY` originates from cassette usage. Because of the relative low reliability of cassette storage, it was advisable to verify the stored program. In this age of affordable disk drives, this function is really superfluous.

The disk drive checks the data it has written to a sector for accuracy. Verify is automatically performed upon every write access by the 1570/71 disk drive. This is also why saving a program takes somewhat more time than loading it.

1.2.6 BLOAD/BSAVE - Saving/loading machine language

BASIC > 3.0:	BLOAD "program name",Dx,Uy,ON Bz,Pa BSAVE "program name",Dx,Uy,ON Bz,Pa TO Pb
Abbreviation:	bL / bS
BASIC <3.0:	LOAD "x:program name",y,1 SAVE not available
optional Parameter: x	
Monitor:	L"x:program name",yy,aaaa S"x:program name",yy,aaaa,bbbb+1
aaaa/bbbb = Start & End address of program	
Parameters (optional):	
Dx: x =	Drive (0/1)
Pa: a =	Starting address (decimal)
Pb: b =	Ending address (decimal)
Uy: y =	Device # (4-15)
Bz: z =	Bank number (0-15)

BLOAD, like the name says, is another load command. But we have already encountered DLOAD. Why do we need another command?

The solution to this puzzle lies in the way in which data are loaded. With DLOAD, the program is always loaded at the start of the BASIC storage, regardless of the area from which it was saved. This isn't bad for BASIC programs. But programs which are written in machine language may not run if loaded with DLOAD. They can execute only in a certain memory area. Graphic pictures too must be loaded to the original location.

For this reason the start address of the program is always saved along with the program itself. BLOAD loads the program back at this address.

The counterpart of BLOAD is BSAVE. This command is used to save arbitrary sections of memory. DSAVE saves only the BASIC program located at the start of the BASIC storage.

The corresponding load and save commands of the monitor require you to specify a memory range for DLOAD. The monitor commands cannot be accessed from a BASIC program. This is the purpose of BLOAD and BSAVE.

These disk drive functions should not be ignored. They can be used to load sprites or graphic pictures into the proper memory locations. They are also used by the machine language programmer, who can load machine language programs more easily. In these applications, you should not deviate from the start address saved along with the program. Whenever possible, specify the parameter Pa. Then you can be certain that the data is loaded in the proper memory location. Otherwise the program may be loaded into an area with contains important parts of your current program--causing the computer to crash.

One more thing is important. While the starting and ending addresses of the memory range must be given in hexadecimal in the monitor, only decimal values are allowed in BASIC. If you want to use hexadecimal specifications, you must use the command DEC(" "). The expression must be enclosed in parentheses.

Furthermore the BSAVE command or the monitor Save command has a peculiarity. The contents of the last specified range address is not saved. So you should always specify the ending address+1.

1.2.7 DIRECTORY/CATALOG - Display the disk contents

BASIC > 3.0:	DIRECTORY Dx ON Uy,"name" CATALOG Dy ON Uy, "name"
Abbreviation:	diR / cA
BASIC <3.0:	LOAD "$x:name",y : LIST
Monitor:	@y,x:$name
Parameters (optional):	
Dx: x =	Drive (0/1)
Uy: y =	Device # (4-15)
name :	Search string for selection of files

We've now saved and loaded programs several times. But what programs are now on the diskette? Under what name was the last program stored? We need to see the contents of the diskette.

To save you from reaching for pencil and paper, the 1570/1571 disk drive automatically keeps a directory of the programs and files stored on the diskette. It may be displayed by using the `CATALOG` or `DIRECTORY` commands. But why are there two commands to perform the same function? This is also probably a sort of tradition (like `DVERIFY`) because both commands were implemented in the BASIC 4.0 of the CBM-8000 series computers. Also, BASIC 7.0 is supposed to be compatible with all previous Commodore dialects.

The parameters are standard except for `name`. If you specify this parameter, you can select certain files to be displayed. This only makes sense with wildcards, of which you learn more in Section 1.3.9. For example, it is possible to list only the entries whose name begins with "a". If the name specification is missing, the entire directory will be printed.

Now to the directory itself. Let's take a look at an example:

```
0 "CHARTPAK-128 B   " LE 2A
5      "BILLBOARD"         PRG
1      "CHPK.CONFIG"       SEQ
11     "CHPK.PM.1526"      PRG
9      "CHPK.PM.C"         PRG
12     "CHPK.PM.E"         PRG
9      "CHPK.PM.OC"        PRG
9      "CHPK.PM.P"         PRG
1      "CHPK.PRD.C5"       PRG
1      "CHPK.PRD.C6"       PRG
1      "CHPK.PRD.EF"       PRG
2      "CHPK.PRD.EJ"       PRG
1      "CHPK.PRD.EM"       PRG
1      "CHPK.PRD.OC"       PRG
1      "CHPK.PRD.PB"       PRG
2      "CHPK.PRINTERS"     SEQ
```

The drive number, disk name, ID, and disk format are displayed in reverse in the title line. The drive for the 1571 is naturally always 0 since it is a single drive. The disk name and the two character ID follow. The identifier "2A" serves only to recognize which diskette format is involved.

Next the contents are listed. First the number of blocks (sectors) is displayed. This gives an indication of how large the program or file is. After this comes the file name and finally the file type. This specification gives information about the type of the entry, whether it is a file, a program, or whatever. The standard file types are listed below:

DEL	= deleted entry
PRG	= program
SEQ	= sequential file
USR	= user file
REL	= relative file

At the end of the listing is the number of sectors (blocks) which are still free on the disk.

1.2.8 SCRATCH - Deleting programs and files

BASIC > 3.0:	SCRATCH "name1,name2,..",Dx,Uy
Abbreviation:	sC
BASIC <3.0:	OPEN 1,y,15,"SX:name1,name2,.."
Monitor:	@y,Sx:name1,name2,..
Parameter:	
name :	Up to five filenames separated by commas
Parameters (optional):	
Dx: x =	Drive (0/1)
Uy: y =	Device # (4-15)

If you've stored a few test programs and now want to erase them you can use the SCRATCH command to do this. It deletes the entry from the directory and releases the blocks occupied by the program or file.

Up to five entries can be deleted at a time. The names of the individual entries are separated by commas.

It's a very short time between pressing the <RETURN> key and erasing the wrong file because you specified the wrong name. BASIC 7.0 asks you to confirm scratches. If you're sure, press <Y>. Any other key terminates the command.

When a program or file is SCRATCHed, it is not really erased or overwritten. Instead it is just flagged as deleted. It's possible to change this flag and thereby recover the file. A disk monitor which writes directly to the tracks and sectors of the diskette can be used for this.

After deleting, the following message is displayed on the screen:

```
01, FILES SCRATCHED, XX, 00
```

The number XX indicates how many files were removed. This is especially important when using wildcards in the name specification.

The SCRATCH command can naturally also be used in a BASIC program. The question "Are your sure?" is asked only in the direct mode, however. This question is omitted in the program. If you want to check the message in the program, you must request it from the disk drive. More about this in the next section.

One further note. Files which are listed with an asterisk (*) in the directory may not be deleted with the SCRATCH command. The save process was interrupted when these files were being saved. Always use the VALIDATE command with "*" files to remove them from the diskette.

1.2.9 DS/DS$/ST - When an error occurs...

BASIC > 3.0:	PRINT DS / PRINT DS$ / PRINT ST
Abbreviation:	? DS / ?DS$ / ? ST
BASIC <3.0:	10 OPEN 1,y,15 20 GET#1,A$:PRINT A$;:IFST<>64 THEN 20 30 CLOSE1 RUN
Monitor:	@y y = Device #(4-15)

"Only he who does not try, makes no errors." Have you ever made an error? Imagine that you want to load a program and forgot to insert a diskette in the drive. What does the disk drive do? Try it out once!

Immediately the red (1570) or green (1571) LED starts to flash on and off. If the disk drive is in the 1570/1571 mode, the LED flashes twice as fast as in the 1541 mode. If the light flashes after power-up, the internal self-test routine found an error in the operating electronics. In this case consult Section 6.3.6 for help.

So that you can determine the cause of the error, the disk drive stores an error message. This can be read via the variables DS and DS$. This is why these variable names may not be used in your programs. The error message can be read only once. After this the flashing LED on the drive goes out. The next time the error message is read, you will get the OK message. In BASIC 7.0 the last read is stored in DS/DS$. The complete message is displayed on the screen with PRINT DS$.

Let's take a look at the construction of such an error message:

```
NN, MESSAGE, TT, SS
```

Every error has a number (NN). The exact cause of the error can be determined through this. Then comes the name of the error, such as "Read Error". The specifications TT and SS stand for track and sector number of the location at which the error occurred. The exact meaning of the error, the causes and possible solutions are listed in Appendix D.

If you use just the variable DS instead of DS$, you get only a number--in this case the error number. This is often helpful when analyzing a message. If a command was executed without error, then the drive returns an "OK" message, which has the number 0. Naturally, the red/green (1570/1571) light does not flash then. In your programs you should always check to see if DS contains zero after disk commands or if an error occurred. This can be done through the following program sequences, for example:

C-64 mode:

```
10 open 1,8,15 : input#1,a,a$,b,c : close 1
20 if a<>0 then print a;a$;b;c : stop
```

for all messages (including scratch)

```
10 open 1,8,15 : input#1,a,a$,b,c : close 1
20 if a>19 then print a;a$;b;c : stop
```

for errors only (ignore scratch)

C-128 mode:

```
10 if ds<>0 then print ds$ : stop
```

for all messages (including scratch)

```
10 if ds>19 then print ds$ : stop
```

for errors only (ignore scratch)

Beside DS and DS$ there is another variable that gives information about the current system condition, the variable ST. Naturally, you may not use this name for other variables either. The term ST comes from "status," and this is exactly the function of ST. This variable gives information about the status, the condition of the input/output system. The fact that this involves mainly the cassette recorder will not be discussed further here. The bits for the cassette are therefore omitted:

Bit	Dec.	Function
0	1	Time-out by write
1	2	Time-out by read
6	64	EOI end of data
7	128	EOT end of blocks

For disk operation, only bits 0 and 1 as well as bits 6 and 7 are of interest. Bit 6 is called EOI, "End Of Information." This recognizes when the last character of a transmission has been sent (see `DS$` for BASIC < 3.0).

Bits 0 and 1 indicate a time-out. If a device which is connected to the serial bus is addressed by the computer, it must answer within a certain time. Otherwise the computer will assume that the device is not ready. If the time span runs out (time-out), these bits are set. The reason for a time-out can lie in the fact that the device is suited only for sending or only for receiving data.

The other possibility would be that the device (such as a disk drive) is not even connected. In this case the signal "EOT" will be set. EOT means "End Of Tape"; it is a cassette status signal which was transported to disk use.

The variable `ST` is correspondingly corrected after every disk operation. If the drive is not connected or is turned off, bit 7 of `ST` will be set. In this case the computer responds immediately with `"Device not present"`. If the previous disk command was in a program, the program will stop--an annoying feature. But it is possible to check in a program if the drive is turned on, as the examples below show. In addition, you can determine if a diskette is present in the drive.

C-64 mode:

```
10 poke 768,185
20 open 1,8,15,"i"
30 poke 768,139
40 if st and 128 then print chr$(19)"Please turn
   on the disk drive":close1:goto10
50 input#1,a:close 1
60 if a<>0 then print chr$(19)"Please insert the
    diskette": goto 10
```

C-128 mode:

```
10 trap30
20 open 1,8,15,"i":goto40
30 if er=5 then print chr$(19)"Please turn on
   the disk drive":close1:goto10
40 close1
50 if ds<>0 then print chr$(19)"Please insert
   the diskette": goto10
```

1.3 Disk drive system commands

1.3.1 The command channel

As you learned in Section 1.2.1, the computer communicates with the disk drive via special channels. Naturally there is a separate data channel for the error messages from the previous section--the command channel.

As the name says, this channel is responsible not only for errors, but also for commands. All disk commands except LOAD/SAVE/OPEN/CLOSE are sent over this channel. Since this is rather complicated (see BASIC < 3.0), there are separate disk commands in BASIC 7.0. These put together the pure BASIC < 3.0 disk commands and send them to the disk drive.

These disk commands always consist of one character, which is an abbreviation for a function, like "s" for scratch. Then follows the drive specification, which is a remnant from the time of the dual drives on the large Commodore computers. The 1570/1571 is a single drive, so the drive number should always be 0 (device 8, drive 0). If you select drive 1 in spite of this, an error message will result. This syntax is not entirely senseless because Commodore is planning a double drive for the C-128 which will be called the 1572. Sometimes, however, the drive specification is required to select a specific function (see CONCAT command).

If additional parameters, like filenames, must be specified, then a colon follows as a separator, followed in turn by the parameters. If the drive specification is omitted the drive always assumes drive number 0. This means that you can normally do away with a drive number specification (0/1) on the 1570/1571. You may not forget the colon, however, if additional data are to be transmitted.

In BASIC versions < 3.0 the disk commands must always be sent to the drive via the command channel (except for LOAD/SAVE). In order to inform the computer that it should set up a certain channel to the disk drive, we use the OPEN command. This opens the channel for operation. The syntax to OPEN the command channel is:

```
OPEN 1,8,15
```

The first digit (logical file number) is an arbitrary number between 1 and 255 with which the channel will be designated. Next follows the device address of the drive, in this case this is the standard device number of eight. The last number specifies the channel number, here number 15 for the command channel. You can find more about the `OPEN` command in Sections 1.2.1, 1.4.1, and 1.5.1.

Now you can send commands to the disk drive via this channel. If, for example, the scratch command is to be executed, then you must send `"s:filename"`. The `PRINT#` command is used for this:

```
PRINT#1,"s:filename"
```

In this command we find the `1` from the `OPEN` command again. Since we assigned a number to the channel with the `OPEN` command, we don't have to give all of the specifications (device address, channel number, etc.) again if we want to send a message to a special channel--the logical file number suffices.

One last detail you should know. Commands to the disk drive may not be longer than 41 characters. The internal buffer storage of the 1570/1571 does not allow more. With very long filenames this sometimes leads to limitations, especially with the `SCRATCH` or `COPY` commands. You cannot `SCRATCH` three files with 15 character names with one `SCRATCH` command. This does not lead to real disadvantages--you will just have to divide a task up into several partial steps.

1.3.2 COLLECT - Organizing a diskette

BASIC > 3.0:	COLLECT Dx ON Uy
Abbreviation:	collE
BASIC <3.0: optional Parameter: x	OPEN 1,y,15,"Vx"
Monitor:	@y,Vx
Parameters (optional):	
Dx: x = Uy: y =	Drive (0/1) Device # (4-15)

The COLLECT commands puts the diskette directory back in order. In detail, it involves the directory and the BAM, the table of free and allocated blocks, called the Block Availability Map.

The COLLECT command first erases the BAM. Then the drive determines the sectors used by each valid file entry. These are designated in the BAM as allocated. Finally the new BAM is written to the diskette. In addition, the COLLECT command removes all invalid entries from the directory. Now just what are invalid entries?

Such files are designated by an asterisk (*). They are created when an OPEN file is not CLOSEd or if a program is saved which is larger than the free space on the diskette. The saving process is then interrupted, an error message displayed and all previously free blocks are allocated--only the COLLECT command will reclaim them.

1.3.3 RENAME - Renaming a file in the directory

BASIC > 3.0:	RENAME "old" TO "new",Dx,Uy
Abbreviation:	reN
BASIC <3.0:	OPEN 1,y,15,"Rx:new=old"
Monitor:	@y,Rx:new=old
Parameters:	
old :	old filename
new :	new filename
Parameters (optional):	
Dx: x =	Drive (0/1)
Uy: y =	Device # (4-15)

With this command you can give an existing directory entry a new name. As you see above in the syntax diagram, this is not very complicated.

Naturally this function is not just suited for beautifying the directory. It is particularly interesting when files are to be processed from programs. The files can all receive the same name. When changes are made, you must save the file under a temporary name and delete the old file. The temporary name is then changed to the name of the old file. Using this method you end up with one file in the end. You can use this technique not only for files but for programs as well. This way you don't get 100 versions of the program on the diskette, just the most current, and always with the same name.

1.3.4 CONCAT - Chaining files

BASIC > 3.0:	CONCAT Dx,"source" TO Dy,"target" ON Uz
Abbreviation:	cO
BASIC <3.0:	OPEN 1,z,15,"Cy:target=y:target,x:source"
Monitor:	@z,Cy:target=y:target,x:source
Parameters:	
target :	file to be appended
source :	file to which target will be attached
Parameters (optional):	
Dx: x =	Drive (0/1)
Dy: y =	Drive (0/1)
Uz: z =	Device # (4-15)

The CONCAT command allows you to chain a file to another one. The data of the source file is appended to the destination file. The source file is not deleted.

This chaining works only with sequential files (SEQ or USR). Programs cannot be combined in this manner.

The CONCAT command is actually a copy function and is therefore a type of copy command. More about this in the next section.

1.3.5 COPY - Copy files

BASIC > 3.0:	COPY Dx,"source" TO Dy,"target" ON Uz
Abbreviation:	coP
BASIC <3.0:	OPEN1,z,15 PRINT#1,"Cx:target=y:source" CLOSE1
Monitor:	@z,Cx:target=y:source
Parameters:	
target :	name of new file
source :	name of old file
Parameters (optional):	
Dx: x =	Drive (0/1)
Dy: y =	Drive (0/1)
Uz: z =	Device # (4-15)

This command copies individual files. This seems intelligent for a double drive, but what will it do on a single drive like the 1570/1571? Naturally, the application possibilities of the COPY command are somewhat limited. There are, however, useful applications of the command. You have already become acquainted with one of these in the previous section in the form of the CONCAT command.

With a single drive the COPY command can be used to chain files. A new file is formed out of two, three, or even four already existing files. The data of the source files are appended to the destination file in the order in which the names of the source files where specified. Only sequential files (SEQ or USR) can be combined in this manner.

Naturally program files can be copied as well. Only one source program is allowed. Programs cannot be combined in this manner. This is a problem which must be solved in the computer.

Copying individual files still makes sense. If manipulations are to be performed on files, then this should often be tried out on duplicates first. With the `COPY` command you can create a copy of the original file.

One aspect of this should not be overlooked: The destination file must have a name which is not present on the disk. But here too there is a special case. If for both the source and destination files you specify the drive number and use a name which already exists on the diskette, these file will be overwritten by the new destination file. The `CONCAT` command works according to this method.

To copy a program from one diskette to another, you need a special copy program. Such a program is contained on the Test/demo diskette under the name `"sd.copy.c64"`. You must load and start this program in the C-64 mode. It has the disadvantages of being very slow and difficult to use.

1.3.6 BACKUP - Duplicating diskettes

BASIC > 3.0:	BACKUP Dx TO Dy,Uz
Abbreviation:	baC
BASIC <3.0:	OPEN1,y,15,"Dy=x"
Monitor:	@z,Dy=x
Parameters:	
Dx: x =	Drive # of source disk (0/1)
Dy: y =	Drive # of target disk (0/1)
Uz: z =	Device # (4-15)

This is the only command of the C-128 or 1570/1571 which cannot be used at all. BACKUP is intended to duplicate entire diskettes. The destination diskette is formatted at the same time. This works only with a dual drive. On a single 1570/1571 drive, this is senseless.

What do you do if you need to copy an entire diskette. You can make backup copies with a special backup programs. A copy program for backups is included in the "DOS SHELL" on the Test/demo diskette.

1.3.7 DCLEAR - Closing all channels

BASIC > 3.0:	DCLEAR Dx ON Uy
Abbreviation:	dclE
BASIC <3.0:	not available
Monitor:	not available
Parameters (optional):	
Dx: x =	Drive # of source disk (0/1)
Uy: y =	Device # (4-15)

This command closes all of the channels to the disk drive. This is an internal function of the computer. The command does not send a "close channel" command (`CLOSE`) to the disk drive. Open files cannot be properly handled in this manner. To do this there is the `DCLOSE` command (see Sections 1.4 and 1.5).

For file applications, `DCLEAR` has little use. You can, however, terminate `CMD` channels to the disk drive with it. These are data channels to which the normal screen output has been redirected to another device with `CMD`. This allows the output to be written to a disk file instead of the screen.

If you use `DCLEAR` in your own programs, you must be sure that the input and output will take place on the standard devices.

1.3.8 BOOT - Starting the CP/M operating system

BASIC 7.0:	BOOT "name",Dx,Uy
Abbreviation:	bO
BASIC <7.0:	not available
Monitor:	G FF88C (Track 0, Sector 1)
Parameters (optional):	
Dx: x =	Drive # of source disk (0/1)
Uy: y =	Device # (4-15)

`BOOT` is a command with a double meaning. If parameters (`name, ...`) are given, then it behaves differently than when these are omitted. Let's first look at the `BOOT` command with parameters.

The most important parameter is the `name`. The computer searches for a machine language program with this name in the directory and loads it in to the memory area specified by the file in the current bank (normally 0). Execution then begins at the starting address. You must ensure that the machine code makes sense at this address or the computer will crash.

If you simply enter `BOOT`, the computer reads sector 0 on track 1. If the first three characters of the sector are `CBM`, then it is an autoboot sector. Otherwise the boot command is ended.

The autoboot sector must contain a set of data and a startup program. This is then responsible for performing additional actions. For a detailed study of the `BOOT` command, see the book *128 Internals* from Abacus. Section 7.7 of that book explains the command in detail. The book also contains the relevant kernal listing.

Byte	Function
0-2	"CBM"- marker for identification
3/4	memory address of further boot sectors
5	bank number for following sectors
6	number of boot sectors still to follow
7 on	text to be outputted after the message "BOOTING" followed by a zero
-	name of program to load after loading the blocks followed by a zero
-	machine language routine that is executed after loading

If the boot message is specified, it is printed on the screen after `"Booting"`. If no message is to be printed, the separator $00 must be placed in byte 7. After this a test is made to see if other boot sectors are to be loaded (byte 6 not equal to 0). If so, the data in bytes 3 to 5 apply.

The boot command loads the program from the diskette whose name is given in the string following the boot message.

Finally, you can write your own boot routine. The program is loaded into the cassette buffer in the computer and executed. The boot routine must be present because the computer will try to execute whatever it finds in the cassette buffer. A system crash will probably be the result. The CP/M system diskette is started with `BOOT`. The boot routine in track 1, sector 0 switches the Z-80 on, which organizes the loading of CP/M Plus.

As you have no doubt noticed, the `BOOT` command is automatically called after every reset or power-up of the computer. If an appropriate diskette is inserted in the drive, the boot sector will be loaded an executed.

1.3.9 Wildcards

Up to now you have had to specify the whole name of the program or file to which a disk command is to refer. Let's assume that you have created various programs, say `Test1`, `Test2`, and so on, and you want to delete all of these. You would have to enter all of the filenames--a rather time-consuming task.

For this reason the disk drive offers the ability to abbreviate names or to address entire groups of names at once. The key characters are the asterisk (`*`) and the question mark (`?`). This is why these two characters cannot be used in filenames. They are called wildcards.

First we'll talk about the (`?`) wildcard character. The question mark is a place holder for an arbitrary character. For example, if you enter `TEST?` in a `SCRATCH` command as a file name, all files whose name begins with `TEST` plus one additional character are deleted. Thus `TEST1` , `TEST2` and `TESTy` are deleted. There is no limit to the number of question marks that may be used. A file name `?????` refers to all files whose name are five character in length.

The second wildcard character is the asterisk (`*`) . If it is entered alone, the first directory entry on the diskette is selected. Entered following a combination of characters, the asterisk represents "I don't care" characters. For example, `a*` selects any file name that begins with the letter `a` and has any characters (or none at all) following.

A third wildcard is the equals sign (`=`) . This selects file types of a particular kind. To do this, the equal sign is appended to a file name specification followed by the first letter of the desired file type. For example, `a*=p` selects the first file whose name begins with `a` and is also a program file.

Here are some examples of the use of wildcards and the equal sign:

`a*`	first entry which starts with "a".
`a*cd`	as above. Everything after the asterisk is ignored.
`a?`	all two-character names starting with "a".
`???*`	all entries with at least 3 characters.
`a*=s`	all sequential files which start with "a".

Wildcards may not be used with all disk functions. In addition, the use of wildcards has different results with different commands. The table below gives information about this.

Command	is wildcard allowed?	file chosen
DLOAD / BLOAD DVERIFY	always	first identified filename
DSAVE / BSAVE	no	new filename
DIRECTORY	always	all identified files
SCRATCH	always	all identified files
RENAME CONCAT / COPY	no	given filenames only

1.4 The sequential file

1.4.1 What is a sequential file?

There are basically two types of data files that can be managed on the 1570/1571. These are sequential and relative files. The chapters on sequential and relative files discuss each in detail.

Data storage on a diskette is comparable to the scrolls of biblical times. Historically, the first form of written record was the papyrus scroll. Information is written on the scroll in a strict start to finish order. To find a given piece of information, you would begin to search through the whole scroll. If you want to add information, you must add it at the end of the scroll. Insertion in the middle of the scroll is not possible. This very simple method of data storage is also found on computers. This concept is referred to as sequential file storage.

For an example, let's set up a file containing names and birth dates. In it we'll store the first and last names of our acquaintances and their birth dates. For example:

Harvey Miller	3/1/1966
Tom Schneider	7/24/1952
Jean Schmidt	9/2/1967

As the name indicates, the data is stored sequentially. In a sequential file, data items are stored one after another. As a consequence these data items must be later read in this same order. If we've written the above data to a sequential file and want to find the birthday of Jean Schmidt we have to skip over the data of Harvey Miller and Tom Schneider until we get to Jean.

How does the computer recognize the end of a name or a birth date? Let's ask how we do that without a computer? Now it's easier, there's a space between the data items. We could program the computer such that it interprets spaces accordingly. But if someone's middle name is ever entered, such as "Harvey James Miller," then our program doesn't work anymore, since we have four items on a line instead of just three.

To overcome this problem we simply use a different character to separate the data. For computer files this is usually the ASCII value 13. You probably recognize this value already. It has the same code as the

<RETURN> key. But it is also the character that causes the cursor to jump to the next line (carriage return). The RETURN character will be called "carriage return" or simply "CR" from now on.

A sequence of characters which is ended with "CR" is called a *data field*. Several data fields together make up a data record. In our case the names and birthdates would be the individual data fields, which together form a data record. The file would then have three data records. The data records can be distinguished by using another separator. Normally this separation is handled by the program logic, however. This means that the program knows exactly how many data fields each data record has and can therefore tell the start and end of a record.

1.4.2 Opening a file

Now we want to put the example from the previous section into practice and build a sequential file. To do this, we must first tell the disk drive what the file is to be called, what type it will be, and so on. You are already familiar with the command required for this--the `OPEN` command. Let's take a look at the syntax of this command:

```
OPEN x,y,z,"a:name,b,c"
```

The parameters `x`, `y`, and `z` are the logical file number, device address, and secondary address, which we discussed in Section 1.2.1. `"a:"` is the drive number, which can always be omitted on the 1570/1571. Next follows the name. The only limitations are that a filename may not exceed 16 characters, and that it may not exist already on the diskette.

Next follow the important parameters for the file management. The `,b` is the file type and `,c` is the file mode. For a sequential file the file type must be `s`. Then comes the operating mode. If the file is to be set up for writing, that is, for the first time, a `w` (for `write`) is specified here.

As you have probably noticed, the BASIC < 3.0 syntax is used above. Naturally, there is also a BASIC 7.0 command which is easier to use. You should be familiar with both commands. BASIC 7.0 sends nothing more to the disk drive than the BASIC 3.0 command string. Now the equivalent BASIC 7.0 command:

```
DOPEN#x,"name",Da,Ux,b
```

As you can see, the z parameter is missing. The secondary address for files must be between 2 and 14. The computer selects this automatically. The device address Ux may also be omitted for drive 8. Dx is also superfluous since the 1570/1571 is a single drive. Thus the command usually reduces itself to:

```
DOPEN#x,"name",b
```

The logical channel number x must be retained, however. This determines in the individual outputs to which file they will go. Use only numbers between 1 and 127. Values between 128 and 255 have the result that a linefeed (ASCII value 10) will be added to the carriage return after each output.

Now back to our example--the birthday file. Open this file with:

```
DOPEN#1,"birthday",w
```

Now the disk drive goes into action. First it checks to see if the name is already on the disk. This would result in a "File Exists Error". This is why it is advisable to check the status variable DS after the DOPEN command. If it is 0, you can be sure that the file was opened successfully.

The red (1570) or green (1571) LED stays lit as long as the file is open. It tells that you may not remove the diskette from the drive in the meantime. The disk drive is now waiting for data from the computer. The light will not go out again until the last active file was been closed again.

1.4.3 Storing data

After the file has been opened successfully, we can now write the individual data fields into the file. The same command which outputs information on the screen is used for this--the `PRINT#` command.

It is modified a bit for the output via data channels, since you must also know over what channel the data are to go. It is therefore called:

```
PRINT#x,"data"
```

The abbreviation for the `PRINT#` is therefore not the question mark and the #, but `pR`. You must note this when entering programs if you make use of the abbreviations. Our birthday file would contain the following command sequence:

```
10 DOPEN#1,"birthday",w                        Open file
20 INPUT "first name ";a$                      Input data
30 INPUT "last name ";b$                       into variables
40 INPUT "birthday ";c$
50 PRINT#1,a$;CHR$(13);b$;CHR$(13);c$          Write data in file
```

As you see, we must separate the individual data fields with a CR, which is created with `CHR$(13)`. But why is it missing after the last data field, the birthdate? Quite simple--the `PRINT#` command sends it itself automatically. With logical channel numbers over 127, it will also send a linefeed character. There are also cases in which this automatic linefeed is not desired. Then you must simply terminate the `PRINT#` line with a semicolon (`;`). The computer then knows not to output a CR.

1.4.4 Closing the sequential file

Once all the data is entered into the file, you may not simply remove the diskette from the drive. The drive must end the sector chaining, count up the blocks used, and note this in the directory. If all of this has been successful, then the directory entry of the file will be designated to indicate that the file has been closed successfully.

This function to do all this is called with the CLOSE command. Its syntax is:

```
DCLOSE#x ON Uy
```

The x is again the logical number of the data channel to the file. The device can be selected with y. This can be omitted if the device address is 8. But even the channel number is not always necessary. The command DCLOSE alone closes all currently open files. A maximum of 10 can be managed on the C-64/C-128 at once. You will hardly be able to use all of these since no more than three sequential files can be processed at once.

In BASIC < 3.0 there is only one command to close a very definite file. But there is a trick here too. If one closes the command channel, then the disk drive will automatically close all other channels and files as well. So simply open the command channel at the beginning of your BASIC < 3.0 programs. When you then close it again, it has the same effect as DCLOSE.

If you forget to close the file, then the data will still not be lost. You will discover how to rescue it in the next section.

1.4.5 Reading from a file

Just storing data by itself isn't terribly interesting. You also want to be able to do something with it. This is why there are more commands for processing the data in files then there are for writing data in files.

To process the data you must open the file again, now for reading, of course. There are two different modes for doing this, one for normal reading and special mode with which you can recreate improperly closed files.

In order to read normal sequential files, the operating mode must be `r` (for `read`). The computer automatically assumes this if no `w` is given. The birthday file is once again available after,

```
DOPEN#1,"birthday"
```

In order to now read the data into the computer, there are two options, the `GET#` command and the `INPUT#` command. The simpler of the two is the `GET#` command, so we will discuss it first.

The command, like its counterpart `PRINT#`, was modified somewhat for file management. You must specify the channel number of the file to be read from. The syntax of the command is:

```
GET#x,a$
```

Just as the normal `GET` command reads a character from the keyboard, here a character is read from the corresponding file. The disk drive starts at the beginning of the file and reads character by character to the end of the file.

You cannot read any arbitrary character in the file in this manner. This shows you one of the disadvantages of sequential files.

Turning back to our example, you could read the individual data fields back in again in the following manner:

```
10 DOPEN#1,"birthday"
20 GET#1,z$:a$=a$+z$:IF z$<>CHR$(13)THEN20
30 GET#1,z$:b$=b$+z$:IF z$<>CHR$(13)THEN30
40 GET#1,z$:c$=c$+z$:IF z$<>CHR$(13)THEN40
```

```
50 PRINT "last name : ";a$
60 PRINT "first name : ";b$
70 PRINT "birthday : ";c$
```

In the program above, characters are read from the diskette until a separator "CR" occurs. All of the character read up to that point (including CR) are then assigned to a string variable, which can then be processed further (a$,b$,c$). The individual data fields can be separated with this method.

How do you find out when the last data field, the last data record has been read? The status variable ST, discussed in Section 1.2.9, is used for this. Bit 6 of this status variable has the value 1 if an EOI signal was transmitted. EOI means End Of Information. The bit therefore tells us when the last character has been sent. The test to see if this bit has the value 1 could look something like this:

```
IF ST AND 64 THEN ...
```

Make sure that you place a space between ST and the AND command or the computer will interpret it as "s TAN d", and we don't need the tangent in our case. The IF command branches when the last character has been sent. If you want to program a loop that will be exited when the last character has been read, that is, branches when EOI is not set, the line must read as follows:

```
IF NOT ST AND 64 THEN ...
```

You must note one thing yet. The GET# command reads everything, including control characters, with one exception, the ASCII value 0. It is not transmitted. If a character is equal to CHR$(0), nothing will be sent. In this case you will get an empty string. You must always keep this behavior of the command in mind when programming. The following step is always recommended:

```
GET#1,a$:IF a$=""THEN a$=CHR$(0)
```

or

```
GET#1,a$:a$=LEFT$(a$+CHR$(0),1)
```

Even with this small advantage, the GET# command is a model pupil in contrast to its colleague, the INPUT# command. There are many special cases and possibilities for error when using INPUT#.

The INPUT command has been adapted for data channels and is now worded:

```
INPUT#1,a$
```

This INPUT# command behaves just like the INPUT command that takes input from the screen. Naturally, the inputs cannot be arbitrarily long. A string cannot accept more than 255 characters. But as you have already determined with normal input, the termination actually comes much earlier. This has to do with the fact that the computer stores all input in a buffer before it processes it. And this is only 88 characters (on the C-64) or 160 characters (on the C-128) long. If more than 88 or 160 characters are read with the INPUT# command, an error message will result. This is, logically, worded "String too long." We must be sure that this error does not occur or the computer will terminate the program. This also leads to the question of how long an INPUT# instruction will actually continue to read characters from a file.

This works just like screen input. When the CR code is sent, which is done when you press the <RETURN> key in normal input, the INPUT command ends the input sequence. The problem lies in the fact that individual data fields may not be longer then 87 or 159 characters and a CR must be at the end. It is the job of the program to make sure of this. You as the programmer must ensure that the data fields do not become longer than this.

But this is not enough. The operating system places other stones in your path. These are the characters ":", ",", and ";". These are normally used in BASIC to separate commands and parameters from each other. The INPUT# command does the same thing when it encounters these characters. If a colon, comma, or semicolon occurs in data field, the INPUT# command behaves as if it had read a CR--it terminates the input and assumes that the data field is done. And as an encore from the operating system you get an "Extra ignored error."

You must pay attention to more than just these three characters. If you read numbers with INPUT#, such as with INPUT#1,a it can lead to more problems. This is always the case when characters other than digits occur in the data field to be read. The computer announces this immediately with "File Data Error." You can prevent this only by making sure that you have stored a data field of the same variable type in which you want to read it in again later.

You have probably asked yourself why there is an INPUT# command at all, given all of these disadvantages. The answer is simple--it is faster than the GET# command. The reason for this is that the disk drive must be readdressed before each message. With the GET# command this happens for every character, while the procedure is required only once per string for the INPUT# command. In spite of this you cannot ignore the GET# command. It is required whenever the INPUT# command would fail.

The second mode for opening a file is the *modify* mode. This mode is used so that improperly closed files can still be read. These are all files which are designated with an asterisk in the directory. In order to rescue such files, one opens them with (for example):

```
OPEN 1,8,2,"file,s,m"
```

As you see, this is again the BASIC < 3.0 command. This is because BASIC 7.0 does not recognize the modify mode.

In order to rescue data from an improperly close file, open a new file, read the data in the modify mode from the unclosed file and write it to the new one and then close this one properly (CLOSE).

The end of an improperly closed file can, like a normal file, be recognized with the status variable ST. There is a problem with this, however. Since the file was not closed, the end marker for sector chaining was also not placed. For this reason you will almost certainly read more data than were actually written. These data come from other sectors which were randomly chained into the sector sequence. The only thing that can be done about this is to process the data manually afterwards.

At the end you must delete this unclosed file. This can be done only with the COLLECT command (Section 1.3.2). The SCRATCH command would free the wrong sectors because of the erroneous sector chaining.

1.4.6 Appending data

Usually just storing data once in a file isn't enough. New data keep arriving that must be added to the sequential file. To do this one must read the entire file into the computer and then store it as a new file. The new data are then placed at the end of the file.

This procedure is very time-consuming. For this reason there is a special disk drive function to append data to a file. If a file is opened with the operating mode "a" (append), data can be added to the file. There is a separate command for this in BASIC 7.0:

```
APPEND#x,"name",Dy,Uz
```

The specifications x, y, and z represent the logical file number, drive, and device number, which you are familiar with from the other commands. The "name" is the name of the file which you want to expand. It can also be given with a wildcard. The disk drive then simply selects the first suitable entry. For our example application, the command would be as follows:

```
APPEND#1,"birthday"
```

All data which will be written to the file, as explained in Section 1.4.3, will be appended to the end of the existing file. The append function can create a problem because when the append function is called the disk drive will assign at least one more block to the file. This happens only for the block specification in the directory. This means that the number of used blocks in the directory does not match the blocks actually allocated to the file. You can determine this by adding up all of the block specifications in the directory; you will get a different value than the directory indicates when you subtract the number of free blocks from the total capacity, 664 blocks.

So the `APPEND#` mode is not completely error free. It must be ended with `CLOSE,` just like the `DOPEN` command, so that the file is properly closed. If you forget this, then not only are the added data lost, but the entire file as well. This also occurs if there is not enough space on the disk for the additional data, the disk drive returns a `"Disk Full"` message.

For this reason you should probably use the `CONCAT` command from Section 1.3.4. This command appends a sequential file to an existing file.

The advantage of using this method lies in the fact that you don't have to touch the old file at all. You store simply store the new data in a temporary file. The method functions exactly as described in Sections 1.4.2 to 1.4.4.

Once the new data are placed in the temporary file, it can be appended to the existing file. In practice this could look like this:

```
10 DOPEN#1,"temp",w
```

... Program section to write data in file ...

```
50 DCLOSE#1
60 CONCAT "temp" TO "birthday"
70 SCRATCH "temp"
```

This method is significantly safer than an `APPEND` procedure. The only disadvantage is that there must be enough space on the disk so that the old file "`birthday`" and the temporary file "`temp`", as well as the new larger file "`birthday`" can be stored on it. The reason for this is that first the new file "`birthday`" is written and then the old file "`birthday`" is erased. Finally, "`temp`" can be removed because the data from "`temp`" are now in "`birthday`".

1.4.7 Using sequential files

As you have already learned in Section 1.4.1, the sequential file is the simplest form of data storage. The data is stored one after the other, that is, sequentially. The data is read back in the same way.

With the CMD command, the normal screen output can be redirected to a data channel. And why shouldn't that be a channel to a sequential file? In this manner you can output program listings to a file. These could then be edited with your word processor, among other things.

The sequential file is always useful when you must store data temporarily or it is not necessary to have free selection in data access. With sequential storage, the entire file must be read until you have found the right entry. This can take a long time for large files.

An alternative would be to first read the entire file into the computer. Then you could access the data as desired, if you have placed them in indexed arrays. In addition, access to variables in the computer runs much faster than reading from the diskette. This area of data processing would exceed the scope of this book. I refer you to the BASIC tutorial books listed in the bibliography for more information.

Storing all the data in the computer's memory has one big disadvantage. As large as the memory may be, it is not inexhaustible. The maximum size of a file would be limited to the memory space in the computer.

1.5 The relative file

1.5.1 What is a relative file?

As you have seen in previous sections, the sequential file is a practical way to store files but is far from optimal. A really capable data form must offer the following characteristics:

ability to select every data record
ability to delete individual records
avoid having to read the entire file
ability to read and write to the file

The 1570/1571 allows you to use the *relative file* .

Let's turn back to our example from Section 1.4.1. There we compared a sequential file to a papyrus scroll. If we extend this example to relative files, then we could compare it to an empty book. You can enter information on any page of this book. The book can be opened to any arbitrary page. In this manner you can access any desired data record. This is just how a relative file behaves.

With a relative file you must define the exact length of each data field, and therefore each data record, before the file is written. To access the 2031st data record, for instance, is not a problem. The disk drive can calculate the position in the file from the fixed data record lengths and the number of the data record. Actually, like the book, the data record is already predefined. Each page can store a very specific set of characters. Normally we don't use the entire page of the book, if you assign each data record its own page.

Now you see the dilemma of data processing. Either we store data sequentially and make optimum use of the disk capacity, since no space is lost between data records, or we define a fixed data record length and some of the space on the diskette is lost. But the advantage gained is that you can select any data record since its possible to calculate its exact position.

Back to a concrete example, the birthday file. As we said, each data record must have a fixed length. To do this you predefine the exact lengths of the individual data fields in the record. The lengths should be chosen such that the data will fit in the field. Naturally there can be cases in which the name, for example, is too long and doesn't fit in the data field. On the average, however the individual data records will never be completely used up. You must select your record lengths between these two extremes. We have done this as follows for our birthday file:

First name	15 characters
Last name	10 characters
Birth date	10 characters
unused	5 characters
total	40 characters

You may have encountered the specification "max. size" in your instruction manual. A relative file may be up to 167132 bytes large--even if the diskette is double-sided. The is a limitation of the disk operating system, which cannot manage larger relative files.

To use the entire capacity of the relative file, we can store a total of 4178 data records of 40 characters each.

1.5.2 Opening a file

We want to establish a relative file. This is no more difficult than for a sequential file. The difference is that you must define the data record length. Values between 2 and 254 characters are allowed. This is set with the operating mode "l" (length). This lets the computer know that it is working with a relative file.

It is no longer important whether a file is opened for read or for writing. This distinction does not apply to relative files. You can overwrite, append, or read records to your heart's content. The DOPEN# command for our birthday file is:

```
DOPEN#1,"birthday",L40
```

An entry is again placed in the directory. If the relative file already exists on the diskette, the length specification "L40" can be omitted. But it can also be specified. It must match the data length that was defined when the file was first opened.

In addition, when you open the relative file you should consider approximately how many data records will be contained in it. Then select the last expected data record and write `CHR$(255)` to it. Through this procedure the disk drive then allocates all previous data records. This process can take up several minutes.

The advantage of doing this lies in that you can be certain that the diskette space will not be used for other storage causing your relative file to run out of room.

1.5.3 Storing data

First we need a command with which we select what data record the write operation will refer. This command is:

```
RECORD#x,y,z
```

In order to be able to access a certain record, the data records have numbers running from 1 to a maximum of 65535. You will hardly need this enormous span. Any relative file whose data records are more than 2 characters long will never reach this maximum--the disk capacity will be exhausted first.

Now on to the parameters of the `RECORD#` command. The number x is the logical file number, exactly the same number as for `PRINT#` or `INPUT#` command. Then comes the record number. Finally, the current position within the data record is specified. A read or write operation would then start at this location. You can use this function to set the position pointer to a data field within the record.

The `PRINT#` command is again available for storing the data. This is used just like is was with sequential files (see Section 1.4.3). The only thing which you have to pay attention to is not to output more data than will fit in the data record. If you try to write beyond the end of the record anyway, the data is ignored and the disk drive will return a `"51 Overflow in Record"` message. Check the error variable `DS` to make sure.

1.5.4 Closing the relative file

In contrast to sequential files, the DCLOSE command is not as crucial with relative files. At least your data isn't immediately lost if you forget the DCLOSE command once.

The sector chaining of the relative file is set up when the file is opened or extended. This cannot be disturbed if the file is not closed.

The blocks used by the file are also always placed in the appropriate table in directory. It is not possible for other files to accidentally overwrite the relative file. For this reason, a relative file is never marked with an asterisk in the directory. It is always fully functional.

In spite of this, you should not omit DCLOSE. It has another function. When the file is closed the disk drive determines the number of blocks allocated and updates the directory entry.

Therefore do not omit DCLOSE. We merely wanted to point out that the relative file is more tolerant of errors.

Beyond this, the disk drive announces not only errors, but also makes available messages when the file is expanded, the record does not exist, the disk is full, and so on.

1.5.5 Changing a record

Data is usually short-lived. For this reason it is important that the data in a file can be changed. This is rather involved for a sequential file.

The relative file does not have any of these limitations in this regard. You may use read and write operations arbitrarily. To change data you need only set the record and position of the change with the RECORD# command. With PRINT# you write over the data field or record.

With the PRINT# command you must under certain circumstances, specify a semicolon after the data. This suppresses the output of CR, which would otherwise be written in the data record.

1.5.6 Appending new records

A relative file can be expanded up to a maximum size of one disk side. You need only access the required data record with RECORD# and write it.

Especially important when expanding is the error message "50 Record not present." When writing, the error message can be ignored (it also arises when first writing to the data record). It signals only that the data record accessed did not exist before and is being constructed.

You may not ignore this error message when reading, however. The disk drive is indicating that an attempt was made to access a data record which is not present.

1.5.7 Finding a record

Searching for data is a troublesome problem. You could easily fill an entire book with this topic. One reason is that there is no optimum solution. So the experts have come up with 1001 ways to order, search through, and manage data. There are many more or less practical management methods. No universal method has yet been found. For this reason we can only begin to look into the problem. If you would like to work with this more intensively, you will find corresponding references in the bibliography.

The main problem with relative files lies in the fact that each data record can be accessed only by the record number. In our example, the birthday file, this method is not terribly useful. You will look either for all persons which have a birthday on a certain date or you would like to find the data of a specific person.

Naturally, you can start assigning numbers to your relatives. This may work for 007 because of the notoriety of the number. But does Aunt Clara have number 102 or 93? Or, when you think of Aunt Clara in the future, do you want to speak only of number 1652. You need then only make sure than when planning the birthday party of 672 to ask 7362 about the well-being of 373 and her daughter 6292...

The problem is clear. Numbering the records is very practical for the computer, the master of juggling numbers. But humans can't do it.

Now to the solution of the problem. Our first thought wasn't so dumb. Each name is assigned a number through the record number. We need a list with the names or the birthdates in which the corresponding record number is assigned. The relative file is basically the same, only it functions in reverse.

In order for the searching to be somewhat efficient, the names should be ordered. So we write a program which reads the names and the corresponding record numbers from the relative file and sorts them alphabetically. This data is then read into the computer each time the file is to be used. Now it is possible to find the desired name quickly. The record number is included so that you can read the remaining information from the diskette file.

Let's look at this somewhat differently. Basically it depends on sorting the records. Only then does the search run fast enough. There are also more refined search methods. In the relative file the data is stored unordered. The idea of sorted names and associated record numbers is really not bad. But then we have an additional file, we need more storage space, storage in the computer, and so on. Although the access time is quite fast, a great deal of time is required for the additional work (sorting, reading the sorted list, etc.).

If we look at it closely, we see that it involves only knowing in what order the data records must be called from the diskette so that they are ordered according to a certain criterion. This criterion can be the name, the date, or any data field in the record. This is called the key. Seen this way, it is not necessary to prepare an ordered list with names and record numbers. It would suffice to sort the record numbers. The first number then corresponds to the alphabetically sorted first name, the second corresponds to the alphabetically sorted second name, and so on. This method always involves creating another file--usually a sequential file. The memory requirements, on disk as well as in the computer, would be smaller since you save the space required for the names.

This has one disadvantage, however. Here it is necessary to read the data in the key file before accessing the information in the relative file.

Since the key file is now sorted, you can select a position in the alphabet (such as the fifth name) get the record number of the name. But basically we only want to know which record is the first, which is the second, and so on. The key file need specify only which record is the next in the alphabet.

In this case we speak of something called a pointer or index. We simply place a number in each data record. This is the record number of the next record in alphabetical sequence. The individual records are then chained in this manner. And what do you place at the end? You can either specify the number of the first record again, or assign it a zero. There is no record 0, so your program can always recognize the end.

Naturally, all of this functions in reverse. You can set up another chain in which the names are sorted in reverse order. This gives you the capability to call the previous record as well as the next record--it doesn't get any easier.

There is one thing we can't do without: the record number at which this chaining begins must be saved. You can, of course, set up a sequential file for this. But it would be more practical to have a file begin at the second or third record. Then you can use the first or second record for such information. This method of organization is the business of your program.

If the relative file is to be chained, you must plan enough room for the chaining pointers in each record. Naturally, one can also chain the file later. To do this, place the pointers in parallel relative file organized in the same manner. This requires little additional storage space. But on the 1570/1571 you can open only one relative file at a time. You must always close the main file, then open the chain file, read the key data, then close the key file again, and open the main file--this is not only a lot of programming work, but is also very time-consuming.

We have now created some nice chains. But what happens if we want to insert a new entry into the file? This is no problem. The new record is simply placed at the end of the file, since the physical location in the file is irrelevant. Now the record must be inserted into the chain correctly. To do this you search for the location after which the record must be inserted in the chain. Then you read the number of the next record. This is entered into the new record. The record after which the insertion will be made receives the pointer value of the new record.

The effort increases considerably with the number of chains, that is, the number of keys. Therefore you must be careful how many and what keys the files is to have.

1.5.8 Using relative files

After all of these theoretical considerations, we want to start putting these things into practice. The topic was a birthday file. We established the length of the key in Section 1.5.1 already.

But this brings up the next problem. We also need space for the chaining pointer. How large is this? A number variable is always stored as a string. That means that it can be between one and five characters long (for a maximum of 65535 records). So there must be space for 4-5 characters, although fewer will often be used.

Here we use a little binary math. Each number between 65535 can be converted into a 2-byte binary number. So we need exactly two bytes per data record for our chaining pointer. The conversion is not that difficult:

```
a = chaining pointer (0..65535)
PRINT#1, CHR$(a and 255) CHR$(a/256)
```

Converting the value back works like this:

```
GET#1,a$ : GET#1,b$
a = ASC(a$+CHR$(0)) + ASC(b$+CHR(0)) * 256
```

Let's assume that the name and the date should be used as keys. You have probably asked yourself how you now sort the file and effect the chaining. If you want to install the chaining on an existing file, this is not terribly simple. For this reason you should establish the keys at the outset. Then you proceed exactly as for appending, which we described in the previous section.

Let us now turn to the individual data records. The record length must be defined--we won't be able to alter that. But this does not mean that you must set the length of the data fields in this manner. This is naturally simpler.

This consideration has much to do with the read commands used. Do you want to use `INPUT#` or `GET#`? With `INPUT#` the data fields must be terminated with CR, which takes up additional room. This has a pay off, however because you can use variable data field lengths very easily with `INPUT#`.

Every data field is then terminated with a CR. When saving the data you must be sure that the field lengths do not exceed 88 or 160 characters. If the field is empty, then you simply save a CR. The advantage is that the problem of data not fitting into a field rarely occurs. There is only the danger that the length of all of the data fields, including the CR characters, which you must not forget, may become longer that the record length. You should calculate the total length in your program and request the user to shorten the input if necessary. The disadvantage of variable fields is that you must read all of the fields from the start of the record in order to reach a given field. With set field lengths you can set the pointer to the current character position with the RECORD# command.

Examples of this topic are found in your disk drive instruction manual. The purpose of this chapter was not to offer you the ultimate solution--there simply isn't one. Instead, we wanted to give you some tips and suggestions for your own programming.

CHAPTER 2

ADVANCED PROGRAMMING

2.1 The direct access commands

2.2 The organization of the diskette

2.3 The organization of files

2.1 The direct access commands

2.1.1 Direct access to individual sectors

The 1570/1571 has a set of powerful commands that let you access the data on a diskette by sector. If you use these types of commands, you will have to perform any data management functions yourself. This is in contrast to the sequential and relative files in which the DOS manages the data for you.

By using the *direct access commands* you can build your own data management system. Of course, you will have to do much more work than is you are using sequential or relative files.

Section 2.1.2 discusses the commands. Sections 2.2 and 2.3 talks about the organization of data on the diskette. By discussing these topics, you may be able to borrow ideas for use in your own data management system.

We advise you to use a new blank diskette before you experiment with the direct access commands. Then you'll avoid the possibility of destroying important data.

From Section 1.1.2, you know that a diskette is organized into tracks and sectors. Since the circumference of the outer tracks is greater than that of the inner tracks, more sectors will fit onto the outer tracks. The tracks are numbered beginning with the outside tracks. Therefore track 1 is the outermost track and contains the most sectors. The innermost track is 35. Theoretically, the 1570/1571 can access up to 40 tracks. These last five tracks are not used in the Commodore formats, however.

A disk formatted in this manner has a capacity of 170K. This is the case with the 1570, for example. The 1571 has two read/write heads and can therefore access both sides of the diskette. As a result, the capacity is also twice as large. But on a double-sided disk there would suddenly be two tracks with the number 1, one on each side. How does the disk drive know what side a given sector is on? To solve this problem the first track on the second side of the diskette is numbered 36 and increases to 70.

The following sector numbers result:

Track	Sector Number
1 - 17	0 - 20
18 - 24	0 - 18
25 - 30	0 - 17
31 - 35	0 - 16
36 - 52	0 - 20
53 - 59	0 - 18
60 - 65	0 - 17
66 - 70	0 - 16

If you can instruct the disk drive to read a sector from the disk, the question arises, what to do with it once it is read? Since the disk rotates at 300 RPM, the individual characters will be read at a speed of 60,000 bytes/second. A BASIC program cannot process such rates of data transfer. The sector must be stored temporarily so that you can process it with normal commands. It is no different for writing a sector.

The disk drive has 4 buffers, each exactly 256 bytes long, the length of a sector. This memory is used when you load programs, work with files, and so on. The following table indicates the number of buffers which each type of file requires:

relative file	3 buffers
load/save program	1-2 buffers
sequential file	1-2 buffers
directory	1 buffer
direct access	1 buffer

Now you can see why two relative files cannot be open at the same time--there are not enough buffers. In the age of cheap memory it is quite rare that a disk drive would have only 1.25K of buffer storage. But this is the case with the 1570/1571.

To be able to access arbitrary sectors, we must reserve a buffer for ourselves. This is also called the direct access method. It involves first setting up a data channel for the direct access. This is done with the following command:

```
OPEN x,y,z,"#a"
```

Theoretically you can also use the BASIC 7.0 command `DOPEN`. But since you must specify the secondary address `z` later, it is more practical to use the BASIC 3.0 command. The secondary address is automatically selected by BASIC 7.0. The parameters `x` and `y` give the channel number and the device address.

A "#" is specified as the filename. This tells the disk drive that a direct access channel should be set up. The 1570/1571 then assigns a buffer to the channel. Its number (0..3) can be specified in "a". Normally you should omit this specification. The disk drive then automatically selects a free buffer. Otherwise you might select a buffer which is already being used for other purposes.

If all buffers or the buffer desired is allocated, the drive returns `"70 No Channel"`. Always check the error variable `DS` after opening the direct access channel.

2.1.2 Block-read and block-write

As we indicated, there are special commands for reading a certain sector into the disk buffer or for writing the buffer to the diskette. The commands are sent over the command channel, channel 15. Therefore for all direct accesses you must first open the command channel (see also Section 1.3.1). The sector commands all have the same format:

```
"aaa:c d t s"
```

The parameters have the following meanings:

`aaa`	Command word
`c`	Channel number (secondary address)
`d`	Drive number (0/1)
`t`	Track number (0..35/70)
`s`	Sector number

The channel number c is the parameter z from opening the direct access channel from the previous section. Now it is certainly clear to you why we wanted to know this parameter. The drive number 1 has no function on the 1570/1571 because they are single drives. In spite of this, you may not omit it. It is always 0. Next follow the data of the desired sector--the track and sector numbers. The individual parameters are separated by spaces or commas in the command string. The command word can be separated from the parameters by a space or a colon.

Now on to the command word. The command word selects the exact disk function (reading/writing). Curiously, there are several commands that perform the the same thing:

read	write
b-r	b-w
u1	u2
ua	ub

The commands `u1/ua` or `u2/ub` are identical. They cause the specified sector to be read into the buffer or to be written from the buffer to the diskette. All bytes of the sector can be accessed in this manner. The commands `"b-r"` and `"b-w"` do the same things, except it is no longer possible to read all of the characters in the sector. This is related to an error in the disk operating system. The instruction manual for the 1570/1571 describes this special feature as a great benefit. But the only time it can be sensibly employed is when you are working with the last sector of a program or a sequential file. In practice this means that you can easily forget about `"b-r"` and `"b-w"`. Nothing more will be said about these commands here.

Now, how do you transfer specific bytes from the buffer to the computer? You do this by using the `GET#` or `INPUT#` command. Usually the `GET#` command is used. `INPUT#` is also possible, of course, a CR must follows a maximum of 87 characters (in 64 mode) or 154 characters (in 128 mode).

The buffer pointer determines the location in the buffer at which the bytes for a read/write command are fetched or written. This is set to the start of the buffer after a `U1/U2` operation. If you want to access a random section of the buffer (sector), you can use the block-pointer command.

The syntax for this is:

```
"b-p c b"
```

The specification c is the channel number (secondary address) which you have specified when opening the direct access channel. With b the position of the buffer pointer can be set. The location b is then the position to which reference will be made upon the next write or read command. Since a sector and therefore a buffer is 256 bytes large, b may have values between 0 and 255.

In programs you normally use variables for the parameters like track and sector. This is no problem. If you output a variable with the PRINT# command, the space necessary to separate the parameters is output automatically. This is actually the sign of the variable value, which is printed as a space for positive values.

Now an example of how you can use the U1/U2 commands:

```
OPEN 1,8,15                  Open command channel
OPEN 2,8,2,"#"               Open access channel
IF ds<>0 THEN PRINT DS$      Buffer free?
INPUT "track ";t             Input track
INPUT "sector ";s            Select sector
PRINT#1,"u1:2 0";t;s         Read sector into buffer
IF ds<>0 THEN PRINT DS$      Sector read properly?
```

Now you may perform data manipulations:

```
PRINT#1,"b-p";2;10           Set buffer pointer
PRINT#2,"new data"           Write in buffer
PRINT#1,"u2:2 0";t;s         Write sector back
IF ds<>0 THEN PRINT DS$      Sector written?
CLOSE 1                      Close channel 1 and 2
```

CLOSE 1 suffices because all other channels are closed when the command channel is closed.

2.1.3 Block-allocate and block-free

The disk drive keeps record of which sectors on the disk are allocated and which are still free. If a sector is designated as allocated, it cannot be overwritten by normal file and program data. The sector commands, on the other hand, do not necessarily follow these rules.

For this reason there are special commands to allocate or release a sector. These are:

`"b-f d t s"` to release
`"b-a d t s"` to allocate

The specification `d` is the drive number (always 0); `t` and `s` are the track and sector numbers of the desired block--simple enough in principle if errors hadn't crept in the disk operating system again. If you specify a sector number over 15 for the `"b-a"` command, not only the sector but the entire track will be allocated. If you allocate a sector which is already allocated, then the command should search for the next free sector. But if it doesn't find one on the track, then it tries the next higher track. But this will be allocated completely. The `"b-f"` command too works only with sector numbers up to 15. In short: Either limit your applications to the sectors 0 to 15, or better yet skip it completely. If you use a separate disk for the data which you manage in direct access, the block management doesn't play a role.

If `"b-a"` and `"b-r"` are to work at all, it is necessary to first initialize the disk with `"i"` (initialize disk).

2.2 The organization of the diskette

2.2.1 The directory

You may have wondered how the directory works. It is kept somewhere on the diskette. So that working with the directory entries is fast enough, these are not scattered wildly but have their own track. On Commodore diskettes this is track 18. Other data cannot be stored on this track. You should not attempt to use direct access commands on track 18. Now it is understandable why only 664/1328 sectors are available for data storage out of a total of 683/1366.

The directory (the directory entries) occupy the sectors 1-18 of track 18 on the first side of the diskette. The sectors are not used in numerical order, but at an interval of 3 sectors. This means data is stored first in sector 1, then in sector 4, then in sector 7, and so on. When the end of the track is reached, the other sectors (2,5,8, etc.) are used in the same manner.

Each sector can store a maximum of eight entries. Therefore you can store up to 144 programs or files on a diskette.

In this and the following sections we will become acquainted with the organization of a directory sector. The first two bytes are the chaining pointer. This indicates the track and sector number of the next directory sector. If no sector follows, then the first byte, which represents the track number, has the value 0. The second indicates how many bytes are contained in the last sector.

Next comes the first entry in the sector, then two unused bytes, then the second entry, and so on.

Byte	Meaning
0	File type Bits 0-3: 0 DEL entry deleted 1 SEQ sequential file 2 PRG program 3 USR user file 4 REL relative file Bit 6: 1= no write access allowed Bit 7: 1= entry closed properly
1/2	Track and sector number of the first data block of the entry.
3-18	Filename of the entry (maximum of 16 characters). The remainder are filled with "shifted spaces" (ASCII value 160).
19/20	Track and sector numbers of the first side-sector block. Used only for relative files.
21	Length of a record. Used only for relative files.
22-25	Unused bytes
26/27	Temporary storage for track and sector number of the first data block of the new file when the current file is overwritten with the "@" function.
28/29	Low and high byte of the number of blocks used by the file. The number is stored in binary form.

One of the most important parts of the file entry is the data type indicator. The abbreviations should be familiar to you from the directory. But what is "DEL"? This indicates that the entry has been deleted. Such entries are not normally listed in the directory. The DOS skips all entries whose data type is 0 when displaying the directory. If you now set bit 7 to 1, an entry with the file type DEL would be listed in the directory, since the file type is no longer 0.

Bit 7 indicates whether the file was closed properly or not. If a new file is placed on the diskette or a new program is stored, the directory entry is first created--also to check to see if an entry with the same name is on the diskette. The file type is also stored, but bit 7 is not yet set. Once the data are saved, the file is closed. The number of sectors used is stored in bytes 28/29 and bit 7 of the file type is set. This makes the entry valid. If bit 7 is not set, an asterisk is printed in front of the file type in the directory. Just setting bit 7 does not correct the error. You should also execute a COLLECT command to put the disk directory back in order. This also guarantees that the file is fully usable again.

Bit 6 has a special function. If it is set, you cannot make any write operations to the entry. This means that the SCRATCH and RENAME command will have no effect. The file (or program) can only be read. Unfortunately there is no BASIC command to set or clear this bit. It can only be done manually using a disk monitor.

2.2.2 The Block availability map - BAM

In the previous sections, several references were made to a table in which entries were made to determine which blocks on the diskette were free and which were allocated. On Commodore diskettes this is called the BAM, an abbreviation for Block Availability Map.

The BAM is stored in sector 0 of the directory track (18). In addition, this sector contains the name you gave the diskette when you formatted it.

Before we concern ourselves with the structure of the BAM, let's take a look at how the sector allocation of a track is stored.

Block	Sectors 0-7	Sectors 8 -15	Sectors 16-23
$12	%11111111	%11111111	%11000000

1 = Sector free 0 = Sector full

Four bytes are present in the BAM for each track. The first byte is a binary value which specifies the number of free blocks. The three bytes following this contain a bit pattern in which one sector correspond to each bit (as long as the sector number exist). If the bit of a sector has the value 1, this means that the sector is still free, available for use. If the bit of a sector has the value 0, then the sector is no longer available for use.

The block specification in the first byte is simply a work saving device for the 1570/1571 operating system. This way the set bits need not be counted each time, since this requires processing time.

You can manipulate the BAM by using a disk monitor such that the number of free blocks on the track does not represent the real state of the allocated sectors. You can for example, enter that a block has 255 free blocks, which is impossible, of course. Since these block specifications are also used for calculating the total number of free blocks which you find listed at the end of the directory, astronomical numbers of over 17,000 free blocks are possible--which in reality, you don't really have at all.

The disk operating system doesn't play along with such games for long. Every time changes are made in the BAM, checks are made to see if the block specifications of the track agree with their bit maps. If a deviation is found, the 1570/1571 responds with `"71 Dir Error."`

The entire BAM is found in sector 0 of the directory track. This sector has the following construction:

Byte	Meaning
0/1	Track and sector numbers of the first directory sector, normally track 18, sector 1.
2	Format designation, always "A" for 4040/1541/1570/1571 (ASCII value 65). The disk is write protected if the format designation is wrong.
3	Bit 7: 0= single-sided 1541/1570 diskette 1= double-sided 1571 diskette
4-143	BAM for disk side 1. Each track is represented by four bytes. The map begins with track 1.
144-159	Disk name, given at formatting. Up to 16 characters. The remainder is filled with "shift + space" (ASCII value 160).
160/161	Two "shift + space" (ASCII 160)
162/163	ID characters of the disk
164	This is supposed to be the version number of the operating system. But this character is always "2", although the 1570/71 uses DOS 3.0.
165	Format designater from byte 3
167-170	Three "shift + space" (ASCII value 160)
171-220	49 zeros
221-255	Number of available sectors per track on the reverse side of the disk. These are the block specifications from the BAM on the other side of the disk. The value from track 36 is in byte 221. The last specification, in byte 255, concerns track 70.

A marker is stored in byte 2 which specifies the format type. If this character is not "A", then the disk drive assumes that another format of directory and BAM management is present. To avoid disturbing this, no write operations are allowed. Such an attempt would be answered with the power-up message, indicating that the operating system can do nothing with this format. Only the direct access commands still function. The sector write command (u2) doesn't bother with BAM or directory entries.

The disk name is placed at byte 145. All characters up to byte 170 are present only for the directory output and give the title line. The contents are completely uninteresting. This puts all of those ID change programs which you see in magazines in a completely different light. If you change the ID here, you only get a different display in the title line. This does not change the actual ID in the header of each sector at all.

2.2.3 Single or double-sided diskettes

You may have asked yourself where the BAM of the second side is "hidden" on a double-sided disk. Sector 0 contains only the data for the first side.

On double-sided disks, the directory track continues on the second side of the disk. The track on the reverse side has the number 53 instead of 18 since the reverse side starts at 36.

The BAM of the second side is found in sector 0 of track 53 from byte 0 to byte 104. This involves only the bit pattern of each track. The byte in which the number of free blocks on the track is still stored in sector 0 on track 18 (see table). Otherwise, the map does not differ from the BAM of the first side.

The remaining sectors of track 53 (numbers 1 to 18) are unused. They can be used for neither the directory nor for files. With the direct access method you can place data in these sectors--perhaps your own disk management map or copy protection.

If a double-sided 1571 diskette is used in a 1541 or a 1570, only the first side can be read. This is causes problems in that the drive terminates the access (loading program, reading data) with the error message "67 Illegal Track or Sector" if the data is partially or completely on the other side of the diskette. If you use the 1571 in the C-64 mode or with a C-64, it behaves just like a 1541. The same problem would then arise. For this reason there are commands which switch to 1571 operation or to the second side (see Section 3.1.4).

2.2.4 Manipulating the directory and BAM

You can, of course, change the format of either the directory of BAM. These manipulations can be divide into two groups. One group, includes such things as format tricks, serving to extend the capabilities of the disk drive or perform other small tasks. The other reduces the directory to chaos. This is used to make it difficult to load certain programs or so that the contents of the diskette can not be listed, and so on.

Naturally, none of these methods will shock the experienced disk programmers. In direct access you can read the BAM and directory sectors and look at what has happened there. A disk monitor shows all such manipulations.

We will avoid an endless list of these tricks here. But we do have some useful ones which make it easier to work with the disk drive.

The first thing to mention would be bit 6 of the file type, by which you can protect individual entries from deletion or overwriting (see Section 2.2.1). Another popular manipulation is changing the filename. This often involves making use of the fact that even with names which are less than 16 character long, all 16 are saved. They are filled with spaces, however.

It is precisely this which we want to change. Perhaps you have also tried to overwrite the block specification of an entry with load command (RUN/DLOAD/LOAD) after listing the directory. This way you don't have to type in the filename again. But this won't function quite correctly. The computer always responds with "Syntax Error." This is not surprising, since, after all, what is it to do with the file type abbreviation which is still in the command line? If you overwrite this with spaces, the whole thing works. But now the effort is almost as great as typing out the filename.

It is our goal that after the filename in the directory, several characters are output that make the line into a completely valid BASIC command line. Then you need only overwrite the block specification and the program will be loaded. You could use the following as the end characters:

`":"`	if `DLOAD` or `RUN` is used
`",8:"`	if `LOAD` is used for BASIC programs
`",8,1"`	for `LOAD` with absolute-loading programs

As you have learned in Section 2.2.1, a shifted-space (ASCII value 160) terminates the filename. It is at this point that the second quotation mark is printed in the directory. This means that your filename may be only up to 14 characters long. In order to perform these manipulations, you don't have to pull a highly complicated disk monitor out of the drawer. These extensions can be easily built-in when saving.

For example:

```
DSAVE ("name"+CHR$(160)+",8:")
```

or

```
DSAVE ("name"+CHR$(160)+":")
```

Here is an example printout of one such directory:

```
0 "1571 PRGS          " AB 2A
2     "USER FILE CREATE" PRG
4     "FORMAT READ":     PRG
4     "FORMAT ANALYZE":  PRG
2     "1571 READER":     PRG
1     "DISK TEST 128":   PRG
651 BLOCKS FREE.

READY.
```

2.3 The organization of files

2.3.1 Programs, sequential and user files

Next we'll discuss how normal programs and files are placed on the diskette. The first two topics are programs and sequential files. In the next section we will say more about relative files, which are more complicated.

The simplest form is still the sequential file. Data items are written one after the other in the file. The information first travels to the buffer inside the disk drive. If the buffer is full, its contents are written to a free sector on the diskette. This must then be designated in the BAM as allocated. When this is done the additional data is handled in the same manner.

This scheme requires that you know which sectors make up the file and in which order you must read them in again. There is a pointer in the directory entry which contains the track and sector numbers of the first data sector (byte 1/2). This tell us where the file begins. So that we can find the next sector and the ones following it, they are chained. The first two bytes of each sector specify the track and sector of the next sector. For this reason a sector can store only 254 bytes of data. This chaining goes on like this until the last sector. This has a 0 as the track number of the next sector. The disk drive recognizes through this that the file ends with this sector.

But normally, not all of the bytes of the last sector are used to store data. For this reason you must also know how many bytes belong to file. This is stored in the second byte of the sector (previously the sector number of the next block).

Sequential files and user files are managed with this chaining method. But what is a "user file"? Actually it is nothing more that a sequential file. They are accessed in precisely the same way as described in Section 1.4. The file type must be "u" instead of "s", however. This gives you the option of selecting between two designations for a sequential file. There is also a disk command which works only with user files (see Section 2.3.3).

Programs are saved in virtually the same manner. The only difference is that the first two bytes of a program file form the start address of the program (low byte/high byte) and are not data. The disk drive does not use this information; the computer uses the start address.

2.3.2 The relative file, the side-sector blocks

The data in relative files are stored no differently from those in a sequential file. But as you know, a relative file is organized in records. You can access any desired record.

The most important thing to do is to define the record length beforehand. This makes it possible to calculate from the record number and the record length the number of bytes which you must skip to reach the desired data record. If you read over all of the previous information in the file, nothing would be gained over a sequential file.

This process is speeded up considerably if you divide the offset (number of bytes to skip) to the desired record by 254. This is exactly the number of bytes which fits into each sector. This means that we can calculate the sector in the chaining sequence in which the record is to be found. The remainder from the division indicates the byte number in the sector at which the desired data begins. Naturally you can now follow the sector chaining in order to find the proper sector. But this would hardly be faster than a sequential file.

The special feature of relative files is that the sector chaining is stored in a special table. This table consists of a maximum of six sectors, which are called side-sector blocks. They are organized as follows:

Byte	Meaning
0/1	Track and sector number of the next side-sector block.
2	Number of this side-sector block (0..5)
3	Length of a record in the relative file
4/5	Track and sector numbers of the first side sector (0)
6/7	Track and sector numbers of the second side sector (1)
8/9	Track and sector numbers of the third side sector (2)
10/11	Track and sector numbers of the fourth side sector (3)
12/13	Track and sector numbers of the fourth side sector (3)
14/15	Track and sector numbers of the fifth side sector (4)
16-255	Track and sector numbers of the data blocks

The important part of the side-sector blocks are the bytes 16-255. Here you'll find a list of the data blocks used. Bytes 16/17 are the track and sector numbers of the first data sector of the file, bytes 18/19 are numbers of the second, and so on. There is room for the track and sector numbers of 120 data blocks in a side-sector block. To be able to form larger files, you simply use additional side-sectors.

But now we'd also like to know where the sector is which contains the desired record. To so this divide the previously calculated number of the blocks to the sector which contains the record. In this manner you can determine in which side sector the specifications for the desired data sector are found. The remainder resulting from the division gives the position of the track and sector specifications in the side sector.

In this way, you now know the sector in which the record is contained. In addition, you can determine the position of the data record in the sector from the remainder of the first division, which we used to calculate the data blocks to the proper sector. Eventually, however part of the data record extends into the next sector. The DOS calculates this from the current position and the record length.

So with the side-sector method you need a maximum of 3 sector accesses, though normally only 1 or 2 sector accesses, until you have found the desired data record. First you read the first side-sector. If you're not lucky, the specifications for the calculated data sector are not contained in this side-sector. This is why each side-sector contains the numbers of the other side sectors (bytes 4-15). Therefore the DOS always knows after the first access, in which side-sector the proper track and sector specifications are contained. Then you must read the side-sector. From this you obtain the position of the data sector. By the third access, at most, the sector with the desired record is found.

But three accesses represents the worst case. Normally one of the side-sectors is always stored in a buffer. Then you know immediately in which side-sector block the desired data sector specifications are found. So two accesses to the disk are usually necessary. If you're lucky and the right side-sector is already in the buffer, or the file is still so small that only one side-sector is needed, you can even read the correct data sector directly. This case is not so rare, since in order to get a file with more that one side-sector, it must be larger than about 30K.

CHAPTER 3

PROGRAMMING THE DISK BUFFERS

3.1 Programs in the DOS buffer

3.1 Programs in the DOS buffer

3.1.1 Memory-read and memory-write

As you read in the preface, the 1570/1571 is controlled by its own microprocessor system. In another section of this book we'll go into these internal matters of the drive more intensively. We'll talk more about programming the disk drive in 6502 assembly language, the language of the built-in processor. But you'll be able to understand the following sections even if you are not an expert assembly language programmer.

As you already know, the disk drive has internal buffer storage. This involves a total of 2K of RAM located in the range from $0000 to $07FF. Part of this RAM is required for system purposes, otherwise the microprocessor could not function. The other part, a total of 5*256 bytes, is used as buffer storage. But more than just data can be placed in these buffers. It is also possible to place programs in 6502 machine language there. These can then be built into the operating system of the disk drive.

Now we need a command to write the program into the disk buffer. The direct access methods would work for this, for instance. In this case you select a special buffer and transfer the program--like data--with the `PRINT#` command. But the 1570/1571 can do even more. There are special commands which serve only to send the contents of certain memory locations of the disk drive RAM to the computer. This command is called "memory-read." Its syntax looks like this:

```
"m-r"+chr$(l)+chr$(h)+chr$(n)
```

`l` = low byte of the memory address
`h` = high byte of the memory address
`n` = number of bytes to be read

The parameters `l` and `h` give the addresses of the desired memory. The parameter `n` is the number of bytes which you want to read. The specification `n` may also be omitted. Then the disk drive assumes that only one byte is desired. The `"m-r"` command will be sent to the disk drive via the command channel. If, for example, you want to read the memory location 151 (hex $97), the command sequence is as follows:

```
a = 151                                    Set address
OPEN 1,8,15                                Open command channel
PRINT#1,"m-r"CHR$(a and 255)CHR$(a/256)
                                           Address to drive
GET#1,a$                                   Byte to drive
PRINT ASC(a$+CHR$(0))                      Output byte value
```

In this program the number of sectors per track of the last IBM-34 format is read.

Byte values, which will then be written to a specific memory location, can be sent to the disk drive as well. The command necessary to this is as follows:

```
"m-w"+CHR$(l)+CHR$(h)+CHR$(n)+CHR$(b1)+CHR$(b2)...
```

As you see, the address is again specified with `l` and `h`. Then follows the number of bytes which will be written at this location in the disk drive RAM. This time you cannot omit the specification `n`. Last come the actual data bytes. A maximum of 34 bytes can be sent with one `"m-w"` command. This is because the input buffer of the 1570/1571 is only 41 characters long. If you want to write larger memory sections into RAM, such as a machine language program, you must write it in several sections.

3.1.2 Memory-execute and block-execute

Just reading a program into the buffer doesn't do anything, of course. You must also be able to start this program somehow. This is done with the `"m-e"` command. The command has the following parameters:

```
"m-e"+chr$(l)+chr$(h)
```

Again a memory location must be divided into low and high bytes. The operating system of the 1570/1571 then jumps to this address. An intelligible program must start at this address or the disk system will crash. When the drive microprocessor encounters the instruction `RTS` in the program, the operating system resumes its work.

The specialists among you now know that one can also call specific subroutines in the disk operating system as well. The following little program, for instance, would destroy a given track completely thereby locking-up your disk drive:

```
10 s = 18
20 OPEN 1,8,15
30 PRINT#1,"m-w"CHR$(0)CHR$(3)CHR$(6)CHR$(32)
    CHR$(163)CHR$(253)CHR$(76)CHR$(160)CHR$(234)  40
PRINT#1,"m-w"CHR$(6)CHR$(0)CHR$(1)CHR$(s)
50 PRINT#1,"m-w"CHR$(0)CHR$(0)CHR$(1)CHR$(224)
60 CLOSE 1
```

The track must be specified in `s`. For this experiment **be sure** to use a newly formatted diskette or a diskette that will not be used any more. This is because the data will not only be completely destroyed, but the operating system will always be rather mixed up by this diskette. The program creates a so-called "killer track", your drive locks-up when the directory is accessed. You must power down the drive to regain control.

You may have noticed that the `"m-e"` command is not used in this example. The program is started through a more refined method via the `"m-w"` command. This should not concern us further. Our intention is to show what possibilities you have with the memory access--even from within BASIC.

If you want to place larger programs in the buffer in order to execute them there, it can take quite some time. The most sensible thing to do would be to read the program from diskette into the buffer and then start it there. You must now combine the `"U1"` and the `"m-e"` commands. The contents of a sector (the program) will be read into the buffer and then executed. The developers of the disk drive decided that this should also be possible with one command. This is called:

```
"b-e c d t s"
"b-e";c;d;t;s
```

The parameters `k` and `l` are the channel and drive numbers, which you have already become acquainted with from the direct access commands. The parameters `t` and `s` are again the track and sector numbers. The selected sector is read into the buffer assigned to the channel. Then a jump is made to the start of the buffer in order to execute its contents as a machine language program.

The command has little advantage over a combination of the "U1" and "m-e" commands. Furthermore, it is seldom used. If you want to read programs from the diskette into the drive's RAM and there execute them, there is another, better command which we will discuss in Section 3.1.5.

3.1.3 The user commands

User commands are those which tell the disk drive to execute programs at certain locations in the memory. The start with a "U" followed by a digit or a letter. This second character selects from among several predefined addresses which can be branched to.

The following user commands exist:

User command	Address	Function
U1 or UA	$CD5F	Block-read command
U2 or UB	$CD97	Block-write command
U3 or UC	$0500	Jump to buffer 2
U4 or UD	$0503	Jump to buffer 2
U5 or UE	$0506	Jump to buffer 2
U6 or UF	$0509	Jump to buffer 2
U7 or UG	$050C	Jump to buffer 2
U8 or UH	$050F	Jump to buffer 2
U9 or UI	$FF01	Switch 1540/41 bus
U: or UJ	$EAA0	Reset
U; or UK	$FE67	Interrupt routine

Some of the user commands jump to buffer 2 (U3-U8). The addresses have an interval of exactly 3 bytes. You can very easily set up a vector table in this buffer. This is a list of jump commands which then branch to the individual functions which are called with the user commands.

The remaining user commands jump to various locations in the operating system. This adds some additional disk drive functions. You are already familiar with U1 and U2 from Section 2.1.

The U9- or UI- command serves to switch between the 1540 and 1541 bus. The 1540 was the disk drive for the VIC-20. Since the VIC-20 had a somewhat higher clock frequency than the C-64, you could make the bus a bit faster with the "UI-" command. The command sequence "UI+" switches the bus back to the 1541 timing. If the + or - is omitted or another character is given, the disk drive will perform a partial reset. The zero page and system pointers will be set up again. The RAM/ROM test is not performed and the drive motor does not run.

The UJ command is the total reset. The 1570/1571 behaves as if you had turned it off and then back on again.

The 1570/1571 contains (in contrast to the 1541) the UK command as well. With this command a jump is made to a BRK instruction (see ROM listing $AA2D). As a result, UK starts the interrupt routine. In normal operation this has no special effect. But if you have inserted your own program in this routine (more about this in DOS chapter), then it can be started in this manner.

The user commands have a powerful advantage over the "m-e" command. You can use them in almost all situations where a program has only a function for entering disk drive commands--whether it is a word processor, database manager, or whatever.

The "m-e" command on the other hand can be used only in BASIC since it needs the CHR$ function in order to transfer the low and high bytes of the start address.

3.1.4 The USER0 commands

It is almost a tradition at Commodore to put many interesting commands in the machines which are not mentioned at all in the instruction manual. And so the 1570/1571 offers a whole set of commands which are responsible for the handling of diskettes in the CP/M format "IBM System 34."

A command number follows all `USER0` commands. This number is composed of various bit data. It is therefore usually inserted into the command chain with the `CHR$` function. Then follow the parameters of the individual commands. All command numbers are composed of the following data:

Bit 0	: drive number (0/1)
Bits 1-3	: Number of the `USER0` function
Bit 4	: Disk side involved 0= side 1 1= side 2
Bits 5-7	: Various control flags

The drive number is always 0 for the 1570/1571, of course. Here the `USER0` commands are already set up for a future double disk drive. On the 1570 bit 4 must also naturally stay at 0 because the 1570 can use only one side of the disk.

All `USER0` commands function only when the disk drive is being used in the 1570/1571 mode. In the 1541 mode they will be ignored. The sole exception to this is command number 31. Here the functions with which one can select disk sides, among other things, are made available.

Let's take a look at these new commands. For all commands the syntax must be:

```
       "U0"+CHR$(31)+"aa"
or     "U0>aa"
```

The appropriate function must be used in place of the characters "`aa`". The following commands have been added:

aa	Function
M1	Switches the disk drive to the 1571 mode. The system will be operated at a 2MHz clock frequency. This allows the C-1571 properties to be used in the C-64 mode.
M0	Switches to the 1541 mode with 1MHz clock frequency.
H0	Activates the head on side 1
H1	Activates the head on side 2 The H command (head) works only in the 1541 mode

Bit 7 controls the disk initialization for M and H:
0= diskette will be initialized after the command
1= diskette will not be initialized after the command
Command number 31 means "with initialization," number 159 "without initialization"

Rx	Sets number of read attempts in zero-page address $6A. The ASCII value x is placed directly in $6A (see zero-page listing for the exact function of the address).
Sx	Sets sector interval for Commodore diskettes ($69)
T	Tests the ROM checksum
x	The ASCII value x will be accepted as the new device address for the disk drive. x must lie in the range 4-15.

Another function, especially important in the C-128 mode, is the file fast-load. As you know, the loading speed is considerably faster on the C-128 than on the C-64. This fast loading is no longer organized via channel 0, but is simply called through a command via the command channel. The data will then be transferred to the computer with the fast bus mode. The command has the following syntax:

```
OPEN 1,8,15,"u0"+CHR$(32)+"filename"
```

Once again bit 7 in the bit pattern (32) controls a special function:

Bit 7: 0= file type will be tested for PRG
1= file type will not be tested
All sequential file types will be loaded

But we want to concern ourselves with the more important USER0 commands. These are the commands for operating the disk drive in the CP/M mode. You must first become acquainted some zero-page addresses. This will be required for programming in machine language, but they can also be used from BASIC.

Address		Function
$3C	60	Logical sector interval for diskettes in the IBM System 34 format. Used for "sector read/write."
$24	36	Header of the last IBM 34 sector.
$5E	94	Bits 0-3 = number of the current error message. This is precisely the value which is normally set in zero-page addresses $00-$05 by the job loop. Bit 7 : 1= diskette is in IBM format 0= diskette is in Commodore format
$60	96	Smallest sector number on the track
$61	97	Largest sector number on the track
$97	141	Number of sectors on the track

For all CP/M functions which support the disk drive, the date will be transferred in the fast bus mode. This transfer mode can be programmed only in machine language, however. BASIC programs are too slow to accept the data. If a CP/M function is called with the appropriate `USER0` command, the disk drive then sends the data, but the computer doesn't receive it.

This is not terribly tragic since the CP/M commands offer the following additional options:

Bit 5:	1= don't read/write sector in buffer
	0= read/write sector from disk to buffer
Bit 6:	1= disregard read/write error
	0= report read/write errors
Bit 7:	1= don't transfer buffer to computer
	0= transfer buffer to computer

Bits 5-7 of the command number control various special functions. The transfer which disrupts things under BASIC can be disabled with bit 7. In this manner a IBM sector is read only into the internal disk buffer. The transfer to the computer can be done with the direct access commands.

To do this you must know that the data of an IBM 34 sector is always stored at address $0300 in the drive memory (buffer 0). The reason for this is that IBM 34 sectors can be composed of up to 1024 bytes and therefore occupy four buffers. This means that you will have to manage four different direct access channels.

Now we come to the question of how the disk drive ascertains what sector length the diskette has. Further, it is possible to write diskettes with different numbers of sectors using CP/M. There must therefore be a way to analyze the diskette in the drive to get the data about the disk format.

The disk drive offers two special functions for this. With one the header of the next sector can be read. An attempt is first made to read an IBM 34 sector. If this fails, the disk drive tests to see if the sector is in the Commodore format. The result of the read attempt is stored in zero-page address $5E. Bit 7 indicates the disk type.

For IBM-34 diskettes, the zero-page addresses $24-$29 can be read, which contain the ID field of the IBM sector. $27, for instance, gives information about the length of the sector.

A second USER0 function yields the additional data of the disk format. This reads all ID fields of an IBM-34 diskette and calculates the following specifications:

1. Command status byte ($5C).
2. Number of sectors on the track ($97)
3. Number of the track in which the header is found.
4. Smallest sector number on the track ($60)
5. Largest sector number on the track ($61)
6. Sector interval.

The specifications are transferred to the computer in the above order in the fast bus mode. A BASIC program would not be capable of receiving the data. In this case you must read them directly from the drive memory with the direct access commands ("m-r").

Command numbers of the analysis commands:

```
Bit 76543210  Function
    000x0100  read next sector header
              x = side number
    y00x1010  analyze track
              x = side number
              y = 1= go to track given as 4th character
                  0= go to track 0
```

The track analysis function cannot be started from BASIC because the USER0 command does not work properly. You should send the following sequence over the command channel:

```
"m-w"chr$(0)chr$(5)chr$(3)chr$(76)chr$(30)chr$(133)
```

The function will be called with "u3". Additional examples of the analysis of foreign formats can be found in Section 4.2.3.

If you have set the disk drive for the IBM-34 diskette in this manner, you can read or write individual sectors with the USER0 direct access commands.

```
Bit 76543210  Function
    abcx0000  read sector
    abcx0010  write sector

    x= side number
    a= transfer buffer to computer
    b= regard errors
    c= read/write buffer
```

As you see, the supplementary function can be specified in bits 5-7 of the sector commands. The desired disk side can be determined with bit 4. On the 1570 drive this bit must always be 0 since this drive can use only one side of the disk.

The parameters of the sector to be processed are sent to the disk drive over the command channel after the command number. The command is then worded:

```
"u0"chr$(command)+chr$(track)+chr$(sector)+chr$(number)+chr$(new)
```

The track and sector numbers must be given as ASCII values. The parameters following allow several sectors to be read one after the other and transferred to the computer, whereby the number of the next sector arises from the sum of the current sector number and sector interval ($3C). This function is useful only if the fast bus mode is being used. Finally, a track number can be specified to which the disk drive will move after the command. In this manner the disk drive can be steered to the next track while the computer is processing the last data.

Finally, the 1570/1571 offers a function which is not possible on many other disk drives--the ability to format different IBM System 34 formats. The syntax of the `USER0` command is:

```
Bit 76543210  Function

    0iyx0110  format IBM 34 diskette

    x= side at which to begin
    y= number of sides to be formatted
       (0= 1 side, 1= 2 sides)
    i= 1= write track index label
       0= don't write index label
```

Parameters:

```
 4th character: Bit 7: 1= IBM System 34 format
                       0= Commodore format
                Bit 6: 1= use specified sector table
                       0= create sector table from first
                          number and interval
                Bits 0-5: smallest sector number on track
 5th character: Sector interval - 1
                For Commodore format: ID1
 6th character: Marker for sector length [1]
                For Commodore format: ID2
 7th character: Last logical track number [39]
 8th character: Largest sector number on track [16]
 9th character: First logical track number [0]
10th character: First physical track number [0]
11th character: Empty byte, filled with the sectors [229]
starting at the 12th character:
                 Here the numbers of the sectors will be
                 listed if bit 6 of the 4th character is
                 set.
```

As you see, the format function is very complex, but it also offers very comprehensive formatting possibilities. There is no format which cannot be created with this `USER0` function. It is even possible to format a disk so that it can no longer be analyzed or read. This is the case if each track contains only one sector, for example, or if all sectors on the track have the same number.

The many possibilities of this command can also be used in BASIC. To do this you should study the BASIC programs in Section 4.2.3, which demonstrate the use of the `USER0` commands in detail.

3.1.5 Autostart files

The autostart files are only mentioned briefly in the 1570/1571 instruction manual. Nothing is said about the function and use of this program form.

An autostart file is a USR file whose contents are loaded into an arbitrary RAM area of the disk drive memory. This means that you need this file form only if you want to execute your programs in the drive memory. Furthermore, autostart files are not as easily constructed as program files. But don't be afraid to use this disk drive function.

Construction of an autostart file

Byte	Function
0/1	Start address in RAM (low byte/high byte)
2	Number of data bytes in this sector (max 255).
3 - n	Data bytes for the autostart program.
n+1	Checksum calculated from byte 1 to byte n.

Autostart programs are organized on the disk like sequential files. The file type must be "USR". The user files are treated just like sequential files with the difference that "u" is given as the file type. You can open user files only with the BASIC 3.0 command since BASIC 7.0 does not support this form of file.

The construction of an autostart file is not very simple. It consists of an arbitrary number of blocks whose structure is represented in the table above. Each of these blocks, which follow one after the other in the user file, is processed separately by the disk drive. Naturally, the user file may also consist of just one autostart block.

The first thing in an autostart block is the start address at which the program data of the block will be stored in drive RAM. Next is the number of bytes which will be occupied starting at this address. The data bytes of the program must be give starting at byte 3. Then follows a checksum, which is calculated by adding the start address, the number of data bytes, and the data bytes themselves together. If a carry results from the addition, it is counted along with the checksum.

In order to operate larger programs in the disk drive memory, they must be divided into sections comprising 255 bytes. A separate autostart block is then created for each of these sections. Since this is rather laborious, we have a program which will do this for you. It creates an autostart file from a program file. The first two bytes of the program file, the start address, are used as the start address of the autostart file.

The autostart file will be loaded by the disk drive and automatically started one you enter:

```
OPEN 1,8,15,"&filename"
```

The program in drive memory will be started at the address of the first autostart block.

```
10 DIM A$(255)
20 INPUT "PROGRAM NAME";B$
30 INPUT "USER NAME";C$
40 OPEN 1,8,0,B$
50 OPEN2,8,2,C$+",U,W"
60 GOSUB 280
70 ON SGN(ST) GOTO 260: A=ASC(D$)
80 GOSUB 280
90 ON SGN(ST) GOTO 260:A=A+ASC(D$)*256
100 PRINT#2,CHR$(A AND 255)CHR$(A/256);
110 P=0
120 FOR N=1TO 255
130 GOSUB 280
140 P=(257*(P+ASC(D$))/256)AND 255
150 IF ST AND 64 THEN 190
160 IF SGN(ST) THEN 260
170 A$(N)=D$
180 NEXT
190 PRINT#2,CHR$(N);
200 FOR M=1 TO N
210 PRINT#2,A$(M);
220 NEXT
230 PRINT#2,CHR$(P);
240 A=A+N
250 ONN/256+1 GOTO 270,100
260 PRINT "ERROR!!"
270 CLOSE2:CLOSE1:END
280 GET#1,D$:D$=LEFT$(D$+CHR$(0),1)
290 RETURN

READY.
```

CHAPTER 4

THE 1570/1571 and CP/M

4.1 How does CP/M control the disk drive?

4.1.1 BDOS and BIOS

If you want to learn more about the CP/M operating system , you'll quickly encounter the terms BDOS and BIOS. The BDOS, an abbreviation for "Basic Disk Operating System," is the part of the operating system which controls working with the disk drive. It is responsible for the management of files, for the organization of the directory and so on. The second part, the BIOS (Basic Input/Output System) is responsible for the physical operation of the disk drive, reading and writing data on the diskette, and so on.

Naturally we cannot publish a complete description of CP/M--not even a basic introduction. This theme is so comprehensive that a book the size of this one could be filled with information. For this reason we will look only at some of the most interesting aspects of disk drive programming under CP/M.

The BDOS is identical on all CP/M systems and manages the data in blocks which comprise 128 bytes. It is responsible only for the logical management and handling of the data. Furthermore, the BDOS is the part of the operating system which offers the programmer a number of functions for operating the disk drive.

The BIOS has the job of reading and writing the data blocks of the BDOS. This part controls the individual drives. For this reason the BIOS is rewritten for each new CP/M system since each computer system is constructed differently. So it depends on the computer system manufacturer, how capable the BIOS is. It can, for example, process several different disk formats, etc.

4.1.2 DPB - Disk Parameter Block

To manage the data, the BDOS must know the exact format of the diskette. It is also important what capacity the diskette has or how many directory entries are possible. In addition, the BIOS must know which tracks of the diskette are are used for data, which for the operating system, and which for the directory. Furthermore, the specifications of the number of sectors per track, sector interval, and so on, are important.

This information is managed in a special table, the DPB (disk parameter block). These specifications are shown in the table on the next page.

The BIOS of the C-128 CP/M+ operating system can process a total of 12 different diskette formats. In addition to the three Commodore formats (C-64, C-128 single-sided, C-128 double-sided), 8 different IBM-34 formats are recognized. The DPB tables are stored in the file CPM+.SYS at address $1980. If you load this file with a debugger like SID or DDT you can see the DPB. When booting the system these tables are placed in the first bank of the C-128 memory along with the BIOS. For this reason it is very difficult to use the second 64K bank for program storage. If you switch to the second bank ($3E or $3F in $FF00), the computer would crash because the program would be overwritten.

The top 8K of the memory is not switched and always contains the upper area of the first 64K bank. You can make only limited use of this area because it is almost completely occupied by the CP/M operating system.

When a diskette is inserted into the drive, the format data for the DPB table can be determined with the BDOS function $1F. To do this, the number of the drive is specified in A (accumulator) and after the call to the BDOS you get the address of the DPB in the HL register pair. The DPB tables of the current drives always lie in the upper 8K block of the bank and can therefore also be called up or manipulated from a program.

```
Byte  Abr. Function
-------------------------------------------------------------
1/2   SPT  Number of 128-byte blocks per track.
-------------------------------------------------------------
2     BSH  Block shift factor
           This number specifies the size of a management
           block of the BDOS. The individual blocks of the
           BDOS are combined into larger entities. The
           following formula applies:
           bytes per management block = 2 ^ (7+BSH)
           The following values result:
           BSH          0   1   2    3    4    5
           Block size  128 256 512 1024 2048 4096 ...
-------------------------------------------------------------
3     BLM  Block mask
           This number specifies the number of 128-byte
           BDOS blocks per management block. The value is
           decremented by one, that is, 7 means that 8
           blocks are contained.
-------------------------------------------------------------
4     EXM  Extend mask
-------------------------------------------------------------
5/6   DSM  Number of 128-byte blocks on the diskette
           (without system tracks) - 1.
-------------------------------------------------------------
7/8   DRM  Number of directory entries - 1.
-------------------------------------------------------------
9     AL0  16-bit allocation map which indicates which
10    AL1  management blocks are used by the directory. The
           first block of the directory track is represented
           by bit 15, the second by bit 14, and so on.
-------------------------------------------------------------
11/12 CKS  Number of directory entries to be checked to
           recognize a diskette change.
-------------------------------------------------------------
13/14 OFF  Number of reserved system tracks.
-------------------------------------------------------------
15    PSH  Marker for the physical size of a sector.
           PSH                0   1   2    3
           Bytes per sector  128 256 512 1024
-------------------------------------------------------------
16    PSM  Number of 128-byte blocks per physical sector-1.
-------------------------------------------------------------
```

4.2 CP/M diskette internals

4.2.1 MFM data recording under CP/M

This section discusses the method in which data is written to the diskette. What interests us is the technique with which the electronics record the data onto the diskette.

This recording process is called MFM. This is an abbreviation for "Modified Frequency Modulation." "Modified" indicates that there is also a normal recording format, called "FM."

First let's talk about the FM process even though the 1570/1571 doesn't use it. This will then make it easier to understand the MFM process.

Most of you know that the read/write head is actually a small coil. This has the property that it functions like a magnet when current flows through it. In addition the polarity, the arrangement of the north and south poles, depends on the polarity of the current. This means that we have a small magnet which we can electronically alternate again and again depending on which voltage polarity is applied.

The diskette consists of a special material that can be magnetized. The magnetic layer then takes on the same magnetic polarity as the coil in the read/write head. By switching the little magnet of the read/write head electronically you can write information on the diskette. Really quite simple! You magnetize the diskette in one direction for all 0-bits and in the other direction for all 1-bits.

If you want to read the data again,the coil in the read/write head is also used. It returns a voltage according to the polarity of the magnetic layer on the diskette. But this happens only when the polarity on the diskette changes. This means that if the entire track on a diskette has the same polarity, nothing happens.

For this reason you can proceed as follows: the polarity of the coil is changed for every 1-bit, but not for a 0-bit. Reading the diskette then gives a short pulse at the read head when a "1" is on the diskette because the polarity on the diskette changed. If this does not happen, then we know a 0-bit is on the diskette.

The drive motor creates a special problem. The recording of a single bit on the diskette is just a few millionths of a millimeter large. If the motor does not run extremely smooth and makes just a tiny start, data is skipped.

If we can send telephone speech to the moon and back, don't we have the technology for somewhat more precise motors? But of course! But do you want to pay several hundred thousand dollars for your diskette drive?

In order to even out the drive fluctuations, you can write clock bits on the diskette. A clock bit always has the value "1" and so creates a pulse at the read head each time. If a pulse occurs, the drive electronics know that they must now expect the data bit. If another pulse is read within a certain time, the data bit is a "1". If this pulse is missing and the next clock bit suddenly appears, then the last data bit must have been a "0".

But how are clock bits distinguished? A bit is a bit, right? Right--the electronics must be told somehow that the next bit is a clock bit. Then some complicated switching is possible to separate the clock and data bits. Section 4.2.2 handles how the electronics automatically recognize the clock bit.

Using the FM process, a byte would look like this:

```
C D C D C D C D C D C D C D C D
1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 0

Date byte:

  0   0   1   0   0   0   1   0

C= clock bite
D= data bit
```

If you think back to Section 1.1.2, you may recall that it wasn't enough to simply write bytes to the diskette, you must also be able to find the stored data again. The problem involved marking the start of a data block, a sector.

This is done with a special marking called the sync character. What does this marker look like? It must be distinguished from the usual recordings. A trick was devised for this: a few of the clock bits are simply omitted. But isn't this dangerous? What happens if the motor speed fluctuates?

Let's pick the value $FE as the data byte. In this value there are a number of data bits with the value "1", which means that there are quite a few pulses on the diskette. In this manner the electronics can find their way when reading and can recognize that the clock bit is missing. We could also interpret the clock bit as a data bit and vice versa. For normal data the clock byte always has the value $FF. For each bit the clock bit is "1". If another clock bit is used, then this data byte can be clearly distinguished from all the other data bytes by means of the clock bits.

Special clock bytes for the FM process:

```
C D C D C D C D C D C D C D C D
-----------------------------------------
1   1   0   0   0   1   1   1     Clock: $C7
  1   1   1   1   1   0   0   0   Data:  $F8
-----------------------------------------
1   1   0   1   0   1   1   1     Clock: $D7
  1   1   1   1   1   1   0   0   Data:  $FC
-----------------------------------------
1   1   0   0   0   1   1   1     Clock: $C7
  1   1   1   1   1   1   1   0   Data:  $FE
-----------------------------------------
```

Normally the data in FM is recorded at a rate of 250,000 bits per second. Naturally you would like to put as much data on the diskette as possible. The first thought would be to increase the recording rate, maybe to 500,000 bits per second. This would double the capacity of the diskette. But there are physical limitations. The magnetic layer is not capable of recording data at this high speed. Since for 500,000 data bits per second, 500,000 clock bits are also recorded, we have a grand total of 1,000,000 pulses per second. It is not possible to write so many pulses since they would overlap each other because they could not be recorded accurately enough for there to be a gap between two "1" pulses.

For this reason we have to try to reduce the number of pulses on the diskette without changing the data rate.

The clock bits are the disrupting factor, since they are not used for data storage but still take up half of the diskette storage. The clock bits are especially important if the data bit has the value "0". In this case we can recognize, with the help of the clock bits, that a data bit is missing. If the data bit has the value "1", the clock and data bits are represented by a pulse, resulting in the high pulse rate. We should then omit the clock bits for all data bits with the value "1" and to write them for 0-bits. With this method there is a sufficiently large interval between individual pulses, which would not be present for successive clock and data bits with the value "1", since the electronics have a certain rise and fall time. The data rate has not changed and is still 250,000 bits per second.

You can say that there is a bit cell present for each data bit on the diskette. If the value of the data bit is "0", a pulse is recorded at the start of the cell, while a "1" bit is represented by a pulse in the middle of the bit cell. For sync and index marks a bit cell of the data byte does not contain a pulse and is thereby identified as a special marker.

4.2.2 The IBM System 34 format

"IBM System 34" refers to a diskette format that is in very widespread use. Almost all disk controller components record the data according to this method. The IBM System 34 format (abbreviated to "IBM-34" from now on), is not the manner in which the data are managed on the diskette, but the method according to which the tracks and sectors are constructed or the sync marks are created, and so on.

In the IBM-34 format, sectors with 128, 256, 512, and 1024 bytes per sector can be used, whereby most diskette formats use sectors comprised of 256 bytes. For this reason we will discuss only the organization of a track with 256 bytes. For other sector sizes the same principle for sector recording is used.

IBM-34 diskettes always use the index hole mentioned in Section 1.1.2. This hole controls the point at which the sector recording is to be begin on the track. When the index pulse is encountered, 80 bytes with the value $4E are recorded on the track. This value is used as the fill value for the gaps when formatting. This gap after the index hole gives the controller time to activate the read/write logic. Then comes the "pre-index", a mark consisting of 12 bytes with the value $00. With this value, pulses are generated on the read/write head for clock bits only. This allows the controller to set its read electronics so that clock and data bits will be separated automatically for normal data bytes. The $00 bytes serve to inform the controller which bits are the clock bits. The marking with $00 bytes is also called "sync", since it synchronizes the controller.

Following the index hole is the "index mark". This tells the controller that previous gap belonged to the index hole, since gaps are also present between the individual sectors. The "index mark" for MFM consists of three bytes with the value $F6, followed by a $FC byte. The clock byte $C2 is used for the value $F6 when formatting. This means that the clock bit is missing between the third and fourth data bits which would normally be required. The controller recognizes the index mark through this since this clock bit is not missing for a data byte with the value $F6.

Farther on in the sector there is a gap with 50 $4E bytes. This gives the controller time to prepare for processing the sectors. Following this gap are 12 bytes with the value $00, representing a sync mark. The next 3 bytes have the data value $F5 and are recorded with the clock byte $A1. Together with the $FE byte they represent the "ID address mark." This mark indicates that the sector header follows. The next six bytes are the sector header.

First the track number of the sector is named. Then comes a byte which specifies the diskette side. The value "0" is used for the front side and the value "1" for the back side of the diskette.

The next byte is the sector number of the data section following the header. The fourth specification is the sector marker, which specifies the size of the data sector. The significance of the byte values follow:

```
00  128 bytes per sector
01  256 bytes per sector
02  512 bytes per sector
03 1024 bytes per sector
```

The sector header is terminated with two checksum bytes, also called CRC bytes.

The sector header is followed by a 22-byte gap with the value $4E, terminated by 12 bytes with the value $00, representing a sync mark.

The "data address mark" follows this, marking the start of the data area. It consists of 3 bytes with the data value $F5 and the clock byte $A1, as well as a byte with the value $FB. Following the data address mark are the 256 bytes of the sector.

Finally, two check sum bytes are stored. These are calculated using the CRC procedure. CRC is an abbreviation for Cyclic Redundancy Check. In this method a polynomial is formed from the individual bits of a data byte.

This polynomial is divided by the generator polynomial, G(x)=X16+X12+X5+1. Normally this division does not come out even and a remainder results. The CRC bytes are the values which you must add to the polynomial of the data bytes so that the division by the generator polynomial does not give a remainder. This sounds complicated, but it is accomplished with simple digital switching.

At the end is another gap of $4E-bytes. The size of this gap depends on the sector size. In addition, larger gaps are used on drives with speed fluctuations which may be up to 3% than are used on more stable drives. Following this gap is the sync mark before the ID address mark of the next sector.

The exact organization of a track can also be gathered from the ROM listing. The routine at $8A86 formats a track in IBM-34 format. From it can be seen which marks are created, how larger the gaps are, and with what parameters the formatting procedure is controlled.

4.2.3 Reading "foreign" diskette formats

One of the best capabilities of the disk drive is its ability to read "foreign" diskette formats. This is used only in the CP/M operating system. The CP/M+ operating system on the C-128 recognizes various formats from Epson, IBM, Kaypro, and Osborne.

If you intend to implement a new diskette format there are several possibilities for doing this. You can add the format data to the BIOS and have CP/M+ recognize the format automatically. Another possibility is to process the diskette format through direct access commands (See Section 3.1.4).

For both applications you must know the exact format of the diskette. These are things like sector length, number of sectors, and so on. These specifications can be determined with the analysis functions described in Section 3.1.4. Since the determination of a recording process is very involved, we present a small BASIC program which does the work for you.

When entering the program be sure to input the spaces and CHR$ codes properly. Default values are specified for the input parameters and you need only press <RETURN> in order to accept a parameter.

The analysis programs first asks for the device address and the drive number. Then the number of the track to be investigated can be specified. Once these inputs have been entered, the analysis begins.

The program first determines if the diskette uses an IBM-34 or a Commodore format. The IBM-34 format is flawlessly recognized by the program. But if neither an IBM format nor a Commodore format is present, (such as if the diskette is unformatted) the program still responds "COMMODORE." Therefore you may view the indication of a Commodore format with some suspicion. Always check in this event if an entire sector or the directory can be read.

Basically, the program serves only to analyze IBM-34 diskettes. Some specifications from the sector header are listed first. These are the track number entered in the sector header, the specification of the diskette side and the sector marker. The last specification indicates how long the sectors on the track are.

Following these are some data which the disk drive has calculated from reading all of the sector ID fields. These are the number of sectors on the track and the smallest and largest sector numbers. In conclusion, all of the sector numbers are listed in the order in which they are located on the track. With this list you can recognize the physical sector interval or spot irregularities in the sector distribution.

Just as exciting as the analysis of foreign disks is the ability to format disks in this format. The following program is used for this:

```
1 dimn(32):bs$=chr$(157):b3$=bs$+bs$+bs$
2 printchr$(14)chr$(147)"IBM System 34 Format
"chr$(17)
3 print"Unit   8"b3$;:input u
4 on1+(u>4)-(u>15)goto2
5 print"Side   2"b3$;:input s:sd=(s/2)and1:a=
sand1
6 printchr$(147)"Number of tracks    40"b3$bs
$;:input nt
7 print"log. Track start    0"b3$;:input tl
8 print"phy. Track start    0"b3$;:input tp
9 print"Sector size         3"b3$;:input si
10 print"Number of sectors   5"b3$;:input sn
11 print"Define sequence (y/n)"
12 geta$:onasc(a$)and3goto13,17:goto12
13 sq=1:fora=1tosn
14 printa;bs$". Sector   "right$(str$(a),2);
15 printb3$;:input n:n(a)=nand31
16 next:goto19
17 sq=0:print "First Sector   1"b3$;:input fs
:fs=fsand31
18 print"Sector skew    0"b3$;:inputsk:sk=((s
k>0)*-sk)and31
19 print"Fill byte     229"b3$bs$bs$;:input b
y
20 b$="u0"+chr$(6+s*16+sd*32)+chr$(128+sq*64+
fs)
21 b$=b$+chr$(sk)+chr$(si)+chr$(nt+tl-1)+chr$
(sn)
22 b$=b$+chr$(tl)+chr$(tp)+chr$(byand255)
23 fora=1tosn:b$=b$+chr$(n(a)):next
24 printchr$(147)"Formatting..."
25 open1,u,15,b$:close1
26 ifds=0then28
27 printchr$(17)chr$(17)chr$(18)"Format Error
"
28 printchr$(17)chr$(17)"one more disk (y/n)"
29 geta$:onasc(a$)and3goto24,2:goto29

ready.
```

```
0 b$=chr$(157):b3$=b$+b$+b$
1 printchr$(147)"Format Analyzer"
2 printchr$(17)chr$(17)"Unit    8"b3$;:input
u
3 on1+(u>4)-(u>15)goto2
4 print"Drive   0"b3$;:input d:d=dand1
5 printchr$(147)"Track   0"b3$;:input tt
6 open1,u,15
7 printchr$(147)"Side 1      :";:s=0:gosub13
8 printchr$(17)"Side 2      :";:s=1:gosub13
9 print#1,"uj"
10 close1
11 printchr$(17)"1 next disk / 2 end"
12 geta$:onval(a$)+1goto12,5:end
13 print#1,"u0"chr$(158)"m1"
14 print#1,"u0"chr$(138+s*16)
15 a=94:gosub34:ifb<128then29
16 print#1,"m-w"chr$(0)chr$(5)chr$(3)chr$(76)
chr$(30)chr$(133)
17 print#1,"u3-"+chr$(tt)
18 print" IBM System 34 format"
19 a=36:gosub34:print"track number:"b
20 a=37:gosub34:print"Side bit     :"b
21 a=39:gosub34:print"Sector size :"b;
22 printtab(17)"("2^(7+b)"Bytes/Sector )"
23 a=151:gosub34:n=b:print"No. of Sec.  :"b
24 a=96:gosub34:print"min. Sector :"b
25 a=97:gosub34:print"max. Sector :"b
26 print"Sequence   :";
27 fora=523to522+n:gosub34:printb;:next:print
28 goto33
29 print" COMMODORE Format"
30 a=24:gosub34:printchr$(17)"Track number:";
b
31 a=22:gosub34:print"ID1 (Dec.)   :";b
32 a=23:gosub34:print"ID2 (Dec.)   :";b
33 return
34 print#1,"m-r"chr$(aand255)chr$(a/256)chr$(
1)
35 get#1,a$:b=asc(a$+chr$(0))
36 return
```

The program allows you to create various IBM System 34 formats. To do this you must specify the device address and drive. Then follow the inputs which determine the format.

The first question concerns the number of tracks to be formatted. The number of the track to be entered in the sector header must be entered. The next input determines at which physical track the formatting will start. This can be used to format only certain tracks, whether for repairing damaged sections or to confuse the controller.

Now the marker for the sector length is required. It may have values between values between 0 and 3. The next question allows the creation of the sequence of sector numbers "by hand." If you don't want to do this, answer with "n".

If no sector sequence is entered, then the program wants to know the number of the first sector and the sector interval. The sector interval is the number of sectors to be constructed between two successive sectors. The program does not check to see if the entered here make sense. The disk drive is the first to determine this and indicates by flashing the error light.

In conclusion you can define a byte with which the sectors will be filled. Normally the value is $E5 (229). Be sure not to enter any values greater than $F0 (250) because these have control functions when formatting.

Now the diskette is formatted. If you answer the question following the formatting with "y", you can create another diskette in the same format without having to re-enter the parameters.

The two BASIC programs are not particularly complex and do not use all of the capabilities of the disk drive. They are intended to show you how diskette programming with IBM-34 diskettes is performed. You may be able to find some suggestions for your own programs in these BASIC programs.

4.2.4 Programming the WD 1770 controller

The control of the IBM-34 recording is performed by a separate controller component in the 1570/1571--the WD 1770 from Western Digital.

In this section we will discuss how this controller is accessed and programmed. This can be done only in machine language and only in the disk drive memory. Additional information about the technical construction of the controller can be found in Section 5.2.4.

The following registers are present for programming the controller:

Address	Read function	Write function
$2000	Status	Command
$2001	Track	Sector
$2002	Sector	Sector
$2003	Data	Data

As you see, register $2000 has different functions when reading and when writing. If a value is written into this memory location, is is interpreted as a command. When reading this address, it doesn't return a command but a value representing the status of the controller. The additional registers serve to pass the command parameters and data to the controller or communicate from the controller to the computer.

The controller recognizes the following commands:

Type	Command	Command value Bit 7 6 5 4 3 2 1 0
1	Restore	0 0 0 0 h v x y
1	Seek	0 0 0 1 h v x y
1	Step	0 0 1 u h v x y
1	Step in	0 1 0 u h v x y
1	Step out	0 1 1 u h v x y
2	Read sector	1 0 0 m h e 0 0
2	Write sector	1 0 1 m h e p a
3	Read address	1 1 0 0 h e 0 0
3	Read track	1 1 1 0 h e 0 0
3	Write track	1 1 1 1 h e p 0
4	Force interrupt	1 1 0 1 i j k l

Meaning of special bits:

```
h:   0= turn motor on, 1= turn motor off
v:   0= verify track, 1= don't verify track
x/y: Step rate 0 0 =  6ms
               0 1 = 12ms
               1 0 = 20ms
               1 1 = 30ms
u:   Set track register to track in sector header
     0= no  1= yes
m:   0= read just one sector
     1= read several sectors
a:   0= set data mark for "sector valid"
     1= set data mark for "sector erased"
e:   0= no head settling time
     1= 30ms head settling time
p:   0= precompensation on
     1= precompensation off
i-l: Interrupt servicing
  i: disregard
  j: disregard
  k: interrupt when index hole encountered
  l: immediate unconditional interrupt
     end command without interrupt for i-l = 0
```

Status register:

```
Bit 0:  Busy flag. Indicates that the command is being
        executed.
Bit 1:  Data request/index
        For all other commands this bit signals that data
        can be taken from register $2003 or can be written
        in the register.
Bit 2:  Lost data/track00
        For commands of type 1 this bit indicates that the
        head is on track 0.
        For all other commands this bit indicates that the
        data in register $2003 was not read or written by
        the program in time.
Bit 3:  CRC error. The checksum bytes of the header or the
        data block decoded an error.
Bit 4:  Record not found. The specified track or sector was
        not found.
Bit 5:  Spin-up/Record type
        For commands of type 1 this bit specifies that 6
        diskette rotations have taken place. For commands of
        type 2 and 3 this bit was the value of the "data
        mark."
Bit 6:  Write protect. This bit indicates when writing that
        the write protect tab is in place.
Bit 7:  Motor on. This bit gives the status of the motor.
        0= motor off  1= motor on
```

As you can see, the controller commands are divided into different command types. The various command types use the status register in different ways and specify which parameter registers are used in a certain manner. Some commands or command bits control the stepper and drive motors. This task is not performed by the IBM-34 controller on the 1570/1571 but by the operating system. Therefore commands of type 1 are meaningless on the Commodore disk drive.

The commands of type 2 write and read individual sectors. Before one of these commands can be passed to the command register, the number of the desired sector must be written to register $2002. If the desired sector is not present, the controller tries five times to find the sector. If this is not successful, then bit 4 in the status register is set. The sector register $2002 indicates the number of the next available sector.

The commands in group 3 serve to process entire tracks and to analyze the track. The first command, the "Read Address" command, reads the next occurring sector header and outputs it via data register $2003. The two CRC bytes are also passed. Status bit 3 indicates if these bytes are correct or if a checksum error occurred.

The "Read Track" command serves to read an entire track, including the address marks, the gap bytes, and so on. The gap bytes may have the wrong values if they are intended to synchronize the controller. An entire track can be read and analyzed with this function.

The opposite is the "Write Track" function, which writes an entire track. This command is used for formattng the track. For this reason not all of the byte values are written as data bytes on the diskette. The values from $F5 to $F7 have special control functions:

```
$F5   ID address mark. Writes $F5 with clock byte $A1
      (missing clock bit between bits 4 and 5)
$F6   Index mark. Writes $F6 with clock byte $C2
      (missing clock bit between bits 3 and 4)
$F7   Writes two CRC to the diskette instead of the byte.
      The checksum is calculated with the data since the
      last address mark.
```

The track functions start reading or writing when the index hole is encountered. The track is processed until the index hole is encountered again and the diskette has made one complete revolution.

Unfortunately it is not possible to copy entire tracks of a diskette to another track or to another diskette with these two commands. The reason is that errors occur with the gap and synchronization bytes when reading. Beyond this, and this is the most serious problem, the data bytes $F5-$F7 are not written as data bytes but are interpreted as control values for address and identification marks.

The interrupt command serves to interrupt the current function. The condition under which the command is interrupted can be set through bits i-l. After this command, you must wait at least 32 microseconds before the controller may receive the next command. Otherwise it will not interrupt the current command.

CHAPTER 5

PROGRAMMING FOR PROFESSIONALS

5.1 How the bytes appear on the diskette

5.1.1 The organization of a sector

The fundamentals of sector organization was discussed in Section 1.1.2. We'll now discuss this topic in more depth.

As we already mentioned, the start of each sector is provided with a special marker on Commodore diskettes. Through this the electronics can recognize the start of a sector on the track. This marker is called a synchronization mark, or "sync" for short.

```
-----------------------------------------
-      -        -     -      -      -     -
- sync - header - gap - sync - data - gap -
-      -        -     -      -      -     -
-----------------------------------------
```

The figure shows the basic structure of a sector. The sector starts with a sync mark. Then follows the sector header, which contains the following:

```
----------------------------------------------------------
-     -          -        -       -     -     -     -     -
- $08 - checksum - sector - track - ID2 - ID1 - $0F - $0F -
-     -          -        -       -     -     -     -     -
----------------------------------------------------------
```

The first byte of the header serves to identify the header and has the value $08. The disk drive determines if a sync mark, a header, or a data field follows. The data section starts with the marker $07.

Next follows the checksum of the header. In order to calculate it the track and sector numbers as well as both ID characters are added. The next two bytes of the header contain track and sector numbers of the header. The disk drives uses this data to find a given sector.

Finally, every sector header contains the two ID characters which were specified when you formatted the diskette. These characters are read and checked upon each access. If the ID characters have changed, then the disk drive assumes that the diskette was changed.

The two byte values $0F have no control function. They produce the bit sequence "01010101" on the disk, which synchronizes the read electronics.

The sector header is followed by a 9-byte gap before the actual data section begins. This gap is to allow enough time to enable and activate the write operation when writing.

Then comes the sector data. In order to recognize the start of the sector exactly, the data section is preceded by a sync mark. The first byte after the sync mark has the value $07. This is so the data section can be distinguished from the sector header. After the data marker follow the 256 data bytes of the sector. At the end is another checksum composed of the sum of the data bytes.

```
---------------------------------------
-      -     -               -        -
- sync - $07 - data bytes - checksum -
-      -     -               -        -
---------------------------------------
```

After each sector is another gap. Its length depends on the number of sectors on the track and the track number.

5.1.2 The sync marks

As you know, the 1570/1571 does not use the index hole to recognize the start of sectors for the Commodore formats but uses specially recorded marks on the disk, called sync marks.

These marks consist of 5 bytes with the value $FF (40 bits with the value 1). The read electronics recognize when more than 10 bits with the value 1 have been read and then generate the sync signal. This signal is used by the disk operating system when it is waiting for the next sector. This mark also tells the read electronics when the data bits of a byte begin by waiting until the 1 signals of the sync mark are past.

This sync mark causes some problems on the 1571. Certainly you have noticed that when booting the CP/M+ operating system, included with the computer, that the disk drive blinks for a long time. In addition, the process runs a good 30 seconds faster if you copy the operating system to a diskette whose second side is unformatted.

The reason for this behavior is that the drive light always flashes when initializing the diskette and the process takes a long time if the sync marks are on the second side. After a new disk is inserted, the 1571 tries to determine if both sides are formatted. To do this it attempts to read from both sides. When reading, the disk drive naturally orients itself according to the sync marks. If you insert a diskette which is formatted on both sides in which you formatted one side of the disk and then flipped it over, the following takes place: The disk drive reads from the reverse side until a sync mark is encountered. If this does not occur within a certain time, then the disk drive assumes that the second side is unformatted. But on the type of diskette described, there are sync marks on both sides of the diskette. The fact that the diskette is running backwards on the second side from the way it is usually, does not make a difference since a sequence of 1-values, the sync marks, has the same effect read forwards or backwards. Only the following data is not the sector header or the data block, but the bytes of a gap.

The read logic therefore signals an error. The catch is that the disk drive then initiates an error-handling procedure. The read is attempted several more times, whereby the head is repositioned slightly. This procedure takes a good deal of time, however.

It is therefore advisable to copy double-sided disks which where used in a single-sided drive by turning them over to a single-sided format. C-64 or VIC-1541 users who want to make better use of their diskettes in this manner must take into account that the initialization process on the 1571 will take somewhat longer.

A solution to the problem would also be possible with the `USER-0` command `"U0>ra"`. This sets the number of read attempts which will be executed for an error to the value 1. This supresses the error routine.

5.1.3 What is GCR coding?

You have probably asked yourself how the data bytes are recorded in Commodore format since data bytes with the value $FF could be interpreted as sync marks.

The recording format is rather exotic because different recording rates are used on different tracks. The Commodore format belongs neither to the single-density formats which transfer data at a rate of 250,000 bits per second, nor to the double-density formats which work at 500,000 bits per second. A recording rate is used which varies between 250,000 to 307,692 bits per second. Since the outer tracks have a greater circumference than the inner tracks, you can also store more data on them. Therefore there are four different track zones on Commodore diskettes:

Track number	Recording rate	Sectors per track
1 - 17	38461 bytes/sec	21
18 - 24	35714 bytes/sec	19
25 - 30	33333 bytes/sec	18
31 - 35	31250 bytes/sec	17

The GCR process is used to record the data. GCR stands for Group Code Recording. In this method, 4 data bits are converted into 5 GCR bits. A data byte, comprised of 8 bits, is represented by 10 GCR bits. To do this one divides the data byte into two halves, the low-order half (0-3) and the high-order half (bits 4-7). The bits of each of these halves are converted according to the table on the next page.

Thse GCR values are chosen such that a zero is written after a maximum of four 1-bits. As a result, after data bytes are converted to GCR bytes, the longest possible sequence of 1's is a sequence of eight, so data will never be interpreted as a sync mark. In addition, no more than two bits with the value 0 ever follow each other with GCR values. This is important because the read electronics equalizes drive fluctuations through the 1-bits.

Data bytes are always converted in groups of four by the disk operating system. In this case the result is exactly 5 bytes with the corresponding GCR values. Data is not converted automatically, however, but must be performed by a program. The DOS contains routines which perform the conversion by means of an algorithm or with the help of tables. The first method has the disadvantage that the program is more complex and runs slowly, while the table method requires more memory space, but is somewhat simpler and faster.

Decimal	Binary byte	GCR code
0	0000	01010
1	0001	01011
2	0010	10010
3	0011	10011
4	0100	01110
5	0101	01111
6	0110	10110
7	0111	10111
8	1000	01001
9	1001	11001
10	1010	11010
11	1011	11011
12	1100	01101
13	1101	11101
14	1110	11110
15	1111	10101

This conversion of the binary data to GCR values and back again is one of reasons the disk drive needs its own microprocessor system and buffer storage. The data cannot be converted as fast as they must written to the diskette (see recording rate). The data are stored temporarily, converted, and then transferred to the disk.

Here are some examples of how 4 binary bytes would be converted to GCR values:

```
Data bytes   $01       $02       $03       $04
Binary value 0000 0001 0000 0010 0000 0011 0000 0100
GCR value    0101001011010100100101010011010100 1110
GCR bytes    $52      $C5      $25      $4C      $4E

Data bytes   $A1       $FC       $65       $9D
Binary value 1010 0001 1111 1100 0110 0101 1001 1101
GCR value    1101001011101010110110110011111100111101
GCR bytes    $D2      $EA      $DB      $3F      $3D
```

5.2 How the bytes get on the diskette

5.2.1 1570/1571 circuitry

The following sections describe the control system of the disk drive. Predominantly this involves how certain electronic components are used in the 1570/1571 and what tasks they perform. Naturally, this section cannot offer a complete introduction into microprocessor techniques. Also, the components used in the 1570/1571 can be discussed only in reference to their functions in the drive.

We will not try to replace a complete schematic here--we can only clarify some of the more important elements of the disk drive.

The heart of the microcomputer is a 6502B processor. This can be driven at a clock rate of 2MHz. The clock can be switched between 1 and 2MHz on the 1570/1571. In the 1541 mode the disk drive uses the slower processor frequency since the VIC-1541 also works with this frequency. If the disk drive is in the 1571 mode, the processer will be clocked at 2MHz. The bus routines are the reasons for the different clock frequencies. In this program sections in depends on the timely course of the bus signals that the disk drive reacts fast enough and that the data are outputted in the proper intervals.

The higher clock frequency of 2MHz has some advantages. The data which are read from the disk can be processed more quickly. This concerns the GCR conversion, for example, since it is now possible to convert a byte from GCR to binary as soon as it is read. Beyond this, the bus can be operated at a maximum transfer rate of 500,000 baud. At this speed it is possible to send an entire track to the computer immediately during reading. The fact that these superb capabilities are not used is the fault of the 1570/71 operating system alone.

Connected to the processor are three input/output components, an IBM-34-format controller, 2K of RAM, and 32K of ROM. These individual components occupy the following address ranges:

Range	Component
\$0000 - \$07FF	2K RAM
\$1800 - \$180F	6522 (VIA1) Controls bus and 1571 electronics
\$1C00 - \$1C0F	6522 (VIA2) Controls recording electronics, motor, etc.
\$2000 - \$2003	WD 1770 Controls IBM-34 recording
\$4000 - \$400F	6526 (CIA1) Controls fast bus mode
\$8000 - \$FFFF	32K ROM Operating system

5.2.2 The interface components

This section involves the interface components of the type 6522 and 6526. The data sheets for the 6522, available from many semiconductor vendors, are recommended for better understanding of these circuits. Unfortunately there is no public support for the 6526 since it is a development of Commodore. Detailed information about the 6526 can be found in the *Anatomy of the C-64* and *C-128 Internals* from Abacus.

We will first talk about the 6522, also called a VIA (Versatile Interface Adapter). The 1570/1571 has two such components. The VIA pins are assigned as follows:

Pin	Name	Function
2-9	PA	8 data lines which can be programmed freely
10-17	PB	8 data lines which can be programmed freely
18	CB1	Control line
19	CB2	Control line
26-33	D7-0	8 data lines to the processor
39	CA1	Control line
40	CA2	Control line

The individual control and data lines of the VIA are controlled by the computer. The VIA has 16 registers which lie in the memory range of the computer, via which the computer can control the input/output component by writing values in the registers. In addition, the VIA has two built-in counters. These count the processor clock pulses. Once the counters have reached certain values, various actions can be generated. This can be used to program a certain time span after which a signal is generated. This is why these counters are usually called timers.

The VIA has two sets of 8 input/output lines. The data register determines which lines are used as input and which as output, whereby each bit of the register corresponds to a line. If the bit has the value 0, the corresponding line is used as input, while the connection functions as input for the value 1. When used as input, the arriving signal is placed in the appropriate bit of the data register. If the data line is switched to output, the level of the corresponding bit in the data register is outputted.

The two data ports are called PA and PB. Port PA has two different data registers. If you work with data register $01, then writing a new value to this register will always affect the control lines. For example, a pulse can be sent over the control line through which the receiving logic recognizes that a new signal is ready on the port. This function is not used by the 1570/1571 however. It is therefore irrelevant which of the two data registers you use.

Register layout of the VIA 6522

```
Address  Function
-------------------------------------------------
n        Data register for PB
n + $01  Data register for PA with handshaking
n + $02  Data direction register for PB
n + $03  Data direction register for PA
n + $04  Low byte of timer 1
n + $05  High byte of timer 1
n + $06  Output value of timer 1 (low byte)
n + $07  Output value of timer 1 (high byte)
n + $08  Low byte of timer 2
n + $09  High byte of timer 2
n + $0A  Serial input/output line
n + $0B  Auxiliary control register
n + $0C  Peripheral control register
n + $0D  Interrupt flag register
n + $0E  Interrupt mask
n + $0F  Data register for PA (without handshaking)
-------------------------------------------------
n= $1800 for VIA1
   $1C00 for VIA2
```

In addition to the input/output lines there are also the control lines CA and CB. These have control functions when writing to the data registers, as mentioned before. CA and CB can also be used as normal input/output lines. This is there task in the 1570/1571. The mode in which the control lines are operated or the level they have is determined in the peripheral control register:

Peripheral control register:

```
Bit    0 : 0   = CA1 input on falling edge
           1   = CA1 input on rising edge
Bits 1-3: 110= CA2 output with low level
          111= CA2 output with high level
Bit    4 : 0   = CB1 input interrupt on falling edge
           1   = CB1 input interrupt on rising edge
Bits 5-7: 110= CB2 output with low level
          111= CB2 output with high level
```

The control lines of the two VIAs are used for the following purposes:

Line	Function
VIA1 CA1	Input for ATN signal of the serial bus Creates interrupt on rising edge of ATN
VIA1 CB1	Write protect signal. Flag in the interrupt register is set on a falling edge. This means that write protect light barrier was interrupted and the disk was changed.
VIA2 CA1	Input. Sets flag in interrupt register on negative edge of the byte ready signal, which indicates that a byte was read or written.
VIA2 CA2	Output for the SOE signal (serial output enable). 1= read/write electronics activated and the byte-ready signal requested.
VIA2 CB2	Head mode 0= write data 1= read data

The control lines are used especially in VIA2. They set the read/write electronics or receive return messages. The most important signal of this type is the byte-ready input. This signal indicates when the read/write logic has processed a byte and written it to the disk or when a byte has been read from the disk and is now available for further processing. The byte-ready signal is also sent to the PA7 input of VIA1 and the SO input (set overflow) on the processor. These last two inputs are used and tested by the disk operating system. In the 1571 mode, PA7 is used, while SO is used in the 1541 mode. A high level on SO has the result that the overflow flag of the processor is set. In this manner the byte-ready signal can be very easily processed with the 6502 instructions BVC and BVS.

The most important tasks of the two VIA components are not accomplished with the control lines, but with the ports PA and PB. Here is the layout of these input/output lines:

```
Line      I/O Function
-----------------------------------------------
VIA1 PB0   I   Data input from the serial bus
VIA1 PB1   O   Data output to the serial bus
VIA1 PB2   I   Clock input from the serial bus
VIA1 PB3   O   Clock output to the serial bus
VIA1 PB4   O   0= ATN will be answered automatically
               1= ATN will not be answered
VIA1 PB5   I   DIP switch 1 (left)
VIA1 PB6   I   DIP switch 2 (right)
VIA1 PB7   I   ATN signal from the serial bus
-----------------------------------------------
VIA1 PA0   I   State of the track 0 light barrier:
               0= head on track 0
               1= head not on track 0
VIA1 PA1   O   1570/71 bus data direction
               0= 1570/71 bus is input
               1= 1570/71 bus is output
VIA1 PA2   O   Active head (only for 1571)
VIA1 PA5   O   Drive mode and processor clock
               0= 1541 mode with 1MHz clock frequency
               1= 1571 mode with 2MHz clock frequency
VIA1 PA7   I   Byte-ready signal
-----------------------------------------------
VIA2 PB0   O   STP1  Second bit of the stepper control
VIA2 PB1   O   STP0  First bit of the stepper control
VIA2 PB2   O   0= drive motor off,  1= drive motor on
VIA2 PB3   O   0= drive indicator (LED) off
               1= drive indicator (LED) on
VIA2 PB4   I   Condition of the write protect
               0= diskette is write protected
               1= diskette is writable
VIA2 PB5   O   DS0
VIA2 PB6   O   DS1
               The signals DS0 and DS1 control the
               recording rate on the diskette.
               (see ROM listing $9409)
VIA2 PB7   I   Sync signal
               1= sync mark encountered
-----------------------------------------------
VIA2 PA    O   Data sent to the write electronics
VIA2 PA    I   Data sent from the read electronics
-----------------------------------------------
```

Another interface component used in the disk drive is the CIA 6526. This input/output device is also used in the C-64 and C-128. The CIA component is very similar to the 6522. It contains a real-time clock with an alarm.

In the 1570/1571 only the serial input/output (SP) and the coresponding clock line (CNT) are used. Both of these inputs are responsible for the transfer of data in the fast bus mode. You can find out more about this in Section 5.2.6.

Naturally, the CIA 6526 also has two 8-bit parallel ports, which are not used. You have the option of using these lines for your own applications.

5.2.3 The WD 1770 controller

The WD 1770 is manufactured by Western Digital and is software compatible with the WD 179x series, which are also produced by other manufacturers. This 28-pin component contains everything necessary for controlling a disk drive. This includes logic for controlling the stepper motor, for example, to move the read/write head. Beyond this, all components required to read and write data are integrated into the WD 1770.

The stepper control of the WD 1770 is not used in the 1570/1571 disk drive. The operating system takes care of this through VIA2. Only the signals for write protect and track 0 are connected to the WD 1770. This inhibits writing to write-protected disks even if the operating software does not check the write protect.

Naturally, it is possible to control the stepper motor through the WD 1770 with some add-on circuitry. If you modify the operating system accordingly, this would have the advantage that the DOS would no longer be concerned with head positioning and could accomplish other tasks instead.

Another possible modification would be to disconnect pin 26. This signal is tied to ground, so the MFM recording procedure is always used. If you use single-density diskettes, recorded with the FM procedure, you can set the controller accordingly by tying pin 26 to 5V (this is possible with a 1K resistor). You can also build a switch with which you can select between double and single density (depending on whether pin 26 is at 0 or 1). The operating software of the 1570/1571 doesn't notice any of this, and the CP/M+ operating system works well with single-density diskettes.

Pin layout of the WD 1770:

```
Pin  Name   Function
--------------------------------------------------
1    CS     Chip select. A low signal on this pin
            addresses the chip.
2    R/W    0= write to registers
            1= read from registers
3/4  A0/1   Address lines which select the desired
            register when CS=0
5-12 D0-7   Data bus to the processor
13   MR     A low level causes a reset
14   GND    Ground connection
15   Vcc    +5V
16   STEP   Output for step pulses for the head
            motor
17   DIRC   Direction
            0= head moves to the outside
            1= head moves to the inside
18   CLK    Input for operating clock of 8MHz
19   RD     Read data. Pulses from the disk. This
            information contains clock as well as
            data bits.
20   MO     Motor on. Switch on output to the motor.
21   WG     Write gate. This output will high if the
            disk is being written.
22   WD     This is the data, together with the
            clock bits, which will be written to the
            disk.
23   TR00   Track 0 input: 0= head on track 0
                           1= head not on track 0
24   IP     Index pulse: 0= index light barrier
                            interrupted
                         1= index light barrier not
                            interrupted
25   WPRT   Write protect:  0= disk is write
                              protected
                            1= disk is writable
26   DDEN   Double density. 0= double density
                            1= single density
27   DRQ    Data request. 1= data register is ready
28   INTRQ  Interrupt request. 1= end command
--------------------------------------------------
```

5.2.4 The Commodore controller

For the Commodore formats the recording is handled by another controller. The word "controller" is going a bit far, since it actually involves a digital logic network to which entire bytes are sent and which writes them in serial to the diskette. This network is placed in a gate array chip produced by Commodore. For this reason we have no description of the pin layout here since Commodore does not offer any support for this device. Those of you who are still interested in the construction of this device should take a look at the older Commodore disk drive models, which did not use this gate array.

The construction of the Commodore hardware has not changed much over the course of the years. For this reason we can refer you to the literature on the 4040 and 1541 since these devices contain the same basic functions as the 1570/1571--just not in a gate array.

The gate array consists of two important parts: a parallel-serial/serial-parallel converter and a BCD counter. The data byte to be written is sent from PA of VIA2 to a shift register. From there it is sent on to the write electronics with the clock CLK, which is created with a programmable divider and the signals DS0 and DS1 (see VIA2), which then amplifies the pulses and controls the head coil.

The same thing happens when reading, only reverse order. In addition, a counter is reset each time a 0-bit is encountered. The counter starts to work when bits with the value 1 are encountered. This happens until a 0-bit is encountered. Once the counter has reached the value 10, the SYNC signal is generated, since no more than 8 ones can occur in a row with normal GCR data.

5.2.5 The 1541 and 1570/1571 modes

The disk drive can be operated in two different modes. The first of these is the 1541 mode and the second is the 1570/1571 mode. In the 1541 mode the 1570/1571 disk drive is intended to be compatible with the 1541. This is why a clock frequency of 1MHz is used in this mode. Primarily the DOS uses the ROM routines $C000 to $FFFF, which are identical to the 1541 ROM. The disk controller routine is not used because the new hardware features, such as the track 0 recognition, are recognized only by the 1571 ROM.

A more serious difference is that only one side of the disk can be used, even double-sided 1571 diskettes. This mode has the advantage that the problems with two-sided disks ("flippies") discussed in Section 5.1.2 do not occur.

The additional functions available via `USER-0` can no longer be performed in the 1541 mode. The only exceptions are the head and mode commands, which do not function in the 1570/1571 mode since they are not allowed there.

The bus can service only the normal C-64 algorithm in the 1541 mode. In the 1570/1571 mode, both the new, fast bus mode and the old bus mode are possible. This depends on the device with which the disk drive is communicating and is determined anew for each transfer.

After the disk drive is turned on it is always in the 1541 mode. The 1571 can be reached only with the `"u0>m1"` command. The C-128 performs this during its reset procedure.

5.2.6 The serial bus - technology and function

The serial bus is the connection between the computer and peripheral devices connected to it. The disk drive is controlled and data are transferred over this bus. It will be thoroughly discussed in this section so that not only can beginners learn how the data communication between the disk drive and computer works, but more advanced programmers can also get useful information about working with the bus.

The bus consists of six lines whose significance is discussed in more detail:

Pin	Name	Function
1	SRQ	Serial request. This line serves as the clock line for the fast bus mode.
2	GND	Ground connection. Sets up a common zero-potential between all connected devices.
3	ATN	Attention signal. Indentifies controller commands.
4	CLK	Clock line in the normal bus mode.
5	Data	Data line in the normal bus mode.
6	Reset	Computer reset signal. If the computer is reset or turned on, all connected peripheral devices will be reset via this line.

The lines GND (pin 2) and Reset (pin 6) cannot be affected by the computer system. GND is the ground potential of the computer. The ground connections of the all the peripherals are tied together through this lien, creating a unified 0V potential because the differences will be equalized. In this manner a logical 0 on all devices corresponds to the same voltage.

The Reset line passes the pulse created on power-up or through the reset button on to the peripheral devices. These then behave as if the result pulse which arises when the peripheral is turned on had been created and enter their initial states. If a device attached to the computer is turned on, no pulse may be created on the line. There are some improperly constructed devices here and there which do exhibit this behavior. This results in the computer being reset if you turn on a device connected to the computer. A tip: first turn on all peripherals you will need and then turn the computer on. This way you can be sure that all devices are in their proper initial states.

The remaining lines of the serial bus are controlled by the computer. The computer is called the controller since it controls the action on the serial bus. Since this communication usually takes place between the computer and the peripheral, this is the normal case, but it is not necessarily so. Two disk drives can exchange information with each other--the bus logic is not confused as a result. Such transfers are not supported by the Commodore operating system, however.

On the serial bus you must note that only one device may send data. This device is called the talker. Several devices may listen at the same time, and these are called listeners. The controller determines which device functions as talker and which as listener.

Now you can understand why two computers cannot be attached to the same peripheral device. In this case there would be two computers which could both issue instructions at the same time regarding who may communicate with whom--resulting in total chaos. In addition, there is a line that may be controlled by only one computer.

The problem of connecting two computers does not lie with the bus logic but with the software, i.e. the operating system. You could develop a program which makes the computer appear like a peripheral device, and could then be attached to another computer system.

A total of four lines are available for transferring data. Since the procedures on the bus can be very complicated, we will use the process of "stepwise refinement" in explaining them. This means that you will learn first about the rough, fundamental relationships. Later we will investigate the processes in more detail--down to the analysis of the schematic of the bus electronics.

The central control of the bus procedures is assumed by the controller. There is a special line, called ATN (attention), which is activated by the controller when it wants to send a command to the peripheral devices. All data sent while ATN is active (high state) are commands. These normally consist of two bytes, whereby the first byte is constructed as follows:

```
Bits 0-4 : Device address of the device concerned
Bit  5   : 1= addressing as listener
Bit  6   : 1= addressing as talker
Bit  7   : 0= marker for primary address (1st command byte)
           1= marker for secondary address
```

Bit 7 of the command byte indicates whether the byte is the first byte, containing the primary address, or the second byte containing the secondary address. When calling a device, the first command byte is sent first. Bits 5 and 6 specify if the device is to act as the talker or a listener. These two bits therefore control whether the connected device will send or receive data. Only one of the two bytes may ever be set since a device cannot send and receive data at the same time.

Bits 0-4 of the primary byte specify the number of the device for which the command is intended, which may lie between 0 and 30. For this reason each device must follow the proceedings on the bus whenever the ATN line is active to see if it is being addressed.

The device address 31 has a special function and serves to set up the data transfer on the bus again. If this address is used, all peirpheral devices reset their bus logic and end the current talker or listener functions. This usually concerns only one device, the one the computer is communicating with. It is also possible, however, for two peripherals to exchange data or for several devices to be in the listener mode. In this case all devices are reset by this device address. For this reset function bits 5 and 6 have the same meanings as for normal device addresses. In this manner listeners and talkers can be reset independently of each other, whereby these commands then receive the designations unlisten and untalk.

In most cases, the addressing of the peripheral device does not suffice. That's because you would usually like to control special functions of the device, which is done through the secondary address. If a secondary address is used, then a second control byte is sent in the ATN command mode. This can be recognized since bit 7 is set. Bits 5 and 6 always have the value 1 for the secondary byte. Bit 4 specifies if the secondary channel is to be opened or closed. The number of the desired channel must be given in bits 0-3.

Here is an overview of the various control bytes:

```
Control byte  Function
--------------------------------------------------
010xxxxx      Talker call
01011111      Untalk
001xxxxx      Listener call
00111111      Unlisten
--------------------------------------------------
1111yyyy      Open secondary address
1110yyyy      Close secondary address
0110zzzz      Send secondary address for listener
              and talker operation
--------------------------------------------------
```

As you already know, every peripheral must analyze the command bytes in the ATN mode to see if it is concerned by the command. What happens if a device is busy? Imagine that the disk drive is formatting a disk or that the printer is outputting a line at this moment. In this phase the processor system of the peripheral device is busy with its work and has no time to pay attention to the bus traffic.

For this reason a procedure called "handshaking" is used, with whose help the data flow can be controlled. Just like you don't go storming into your boss's office but knock on the door first and wait for "Come in," so the bytes are not outputted on the until the computer makes sure that all the devices are listening.

The control signals necessary for this are sent over the clock and data line and have the following significance:

```
Data  0= all peripherals ready
      1= peripheral(s) not ready
Clock 0= controller ready
      1= controller is not sending data
```

The following happens if a device is to be addressed:

First the ATN line is activated, which signals that a command follows. The controller then sets the clock line to the value 1 to indicate that the data byte has not been sent yet. At the same time it places the value 0 on the data line. Through electronic switching, the data line is automatically set to the condition "1" at every peripheral device.

Now the peripheral devices have time to prepare for the reception of the control byte. If the peripheral is ready to process the control byte, then it sets its data output back to the value 0. Once the last peripheral has reset its data line, the level on the data line goes back to zero, which tells the controller that all devices are ready.

The controller determines if anything is even connected at the start of this addressing phase. It checks to see if the data line is set to the value 1 within one millisecond. This gives a peripheral device which doesn't have automatic switching to set the data line enough time to react to the controller call. If the data line is not set high, then the computer outputs the error message "Device not present."

When all peripheral devices are ready, the controller sets the clock like to the 0 state. This is the signal for the connected devices that the transfer is beginning. No more handshaking is done. The peripheral devices must all be fast enough to process the data. A data bit is outputted on the data line with each low pulse on the clock line. In the time span between the two data bits the clock line assumes the condition 1, telling the connected devices that they should indicate whether or not they received the data bit. To do this, the talker, which in this case is the computer, sets the data line to the value 0. If the peripheral device received the data bit, it must tell the talker this by outputting a high level on the data line. The talker notices only to the first device which responds in this manner. If additional listeners are present, the messages of these devices will be ignored. For the talker, it is interesting only whether at least one listener is present or not. If the acknowledgement of the data bit does not suceed, a "Time out" error message results in the status byte of the computer.

The fundamental proceedings on the serial bus are not very complicated, but you are still in no position to develop your own bus control routines, expecially in 6502 machine language. You must first become acquainted with the hardware of the serial bus.

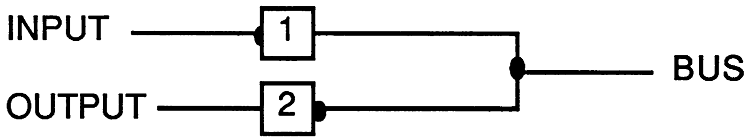
If you use the operating system routines, the bus programming is not difficult. There is a separate routine present for each bus function, like talk or untalk, which need only be called in machine language. The parameters for the bus call must be passed in some zero-page locations and the processor registers.

Here is a list of the most important operating system routines, which are identical on the VIC-20, C-64, and C-128:

```
Name       Address  Function/parameters
---------------------------------------------------
Parameters in zero-page addresses which must always
be set:

$B8        Logical file number
$BA        Device address
$B9        Secondary address + control bits 4-7
$BB/$BC    Address of the filename for OPEN
$B7        Length of the filename
---------------------------------------------------
OPEN       $FFC0  Open data channel (as in BASIC)
CLOSE      $FFC3  Close data channel (as in BASIC)
CKOUT      $FFC9  Output character in A on the bus
CHKIN      $FFC6  Get character from bus into A
TALK       $FFB4  Call talker function
LISTEN     $FFB1  Call listener function
SECTALK    $FF96  Send secondary address after talk
SECLISTEN  $FF93  Send secondary address after listen
UNTALK     $FFAB  Send untalk command
UNLISTEN   $FFAE  Send unlisten command
---------------------------------------------------
```

Assembly language programmers among you will not want to leave anything to the operating system but will want to program everything yourselves. You have probably already studied the layout of the input/output components in the disk drive and the computer.



Notice that two different bits of the input/output port are used for the same bus line. One bit serves for outputting the data and the other is set up as an input. In order to understand what lies behind this peculiar set-up, we must consult the schematic of the bus logic.

As you see, the two connections of the I/O component are connected to the bus line via inverters. The inverter always outputs the precise opposite of the signal fed into it. In this manner the levels on the serial bus always have the opposite value as the bit in the input/output register.

The reason for this is technical in nature. It is not physically possible to set a line to the 0 level and then output the value 1 on the line through another output, such as the peripheral device. The line would never assume the value 1 because the low level would function as a short-circuit.

The entire handling of the signals must therefore be inverted. If the line is to have the value 0, the level 1 is outputted. This remains until some other device set the level 0. This creates a short circuit and the voltage on the line collapses. This then corresponds to the low level.

The devices which are connected to the serial bus always have one output line and one input line. A specific level is placed on the bus through the output line. If this is a "1" level, so that the value 0 is outputted on the bus (remember the inverter), it may occur that this is brought to the 0 state by another device. For this reason each device has an input line with which the actual bus level can be tested.

Basically this special hardware organization is not interesting for programming, because the proper logical values will always be electronically restored by the inverters. But unfortunately there is an exception. This is the input at the computer, the C-64 or C-128. This input has no inverter.

This must be taken into account when programming. If the values of the bus inputs at the computer are to be read, they must always be inverted by the program. If you are waiting for a certain level on a bus line, then you must remember that you must actually wait for the opposite value.

The fact that the physical level is inverted in contrast to the logical value always leads to confusion, especially in the ROM listings. Therefore always look first for the routines which set the data or clock lines to a certain value. You can then recognize if setting a line to the value 1 is commented with the physical or logical level. In the DOS listing in this book the logical level is always commented. Special care is urged with the ROM listings for the C-64 or C-128. There it is usually overlooked that the inputs are not inverted and so lie at the physical bus levels. This leads to comment confusion which no longer has anything to do with the actual proceedings on the serial bus.

As you know, the 1570/1571 can be operated in the fast bus mode. The SRQ line is needed for this mode. This line carries the clock signals which the CIA 6526 creates when outputting a data byte. The data register of the receiving device is controlled by these clock signals.

The SRQ line is also used to determine if the peripheral device concerned can be accessed in the fast bus mode. To do this the calling device sends eight clock pulses on the SRQ line when the level on the data line is reset, indicating that the receiver's readiness. These pulses set a flag in the interrupt register of the computer. When the controller recognizes the resetting of the data line, it checks at the same time to see if the flag in the interrupt register is set. The bus controller then knows that the device can be operated in the fast bus mode. In this case the controller sends eight clock pulses over the SRQ line when setting the clock signal. This tells the peripheral device that it is supposed to send or receive data in the fast bus mode.

In the fast mode, the data bits and clock bits are not inverted, because they are not processed by the CIA. Only the actual data transfer is performed in the fast bus mode. A byte is transferred in exactly 64 microseconds, which would make a transfer rate of 15,625 bytes per second possible.

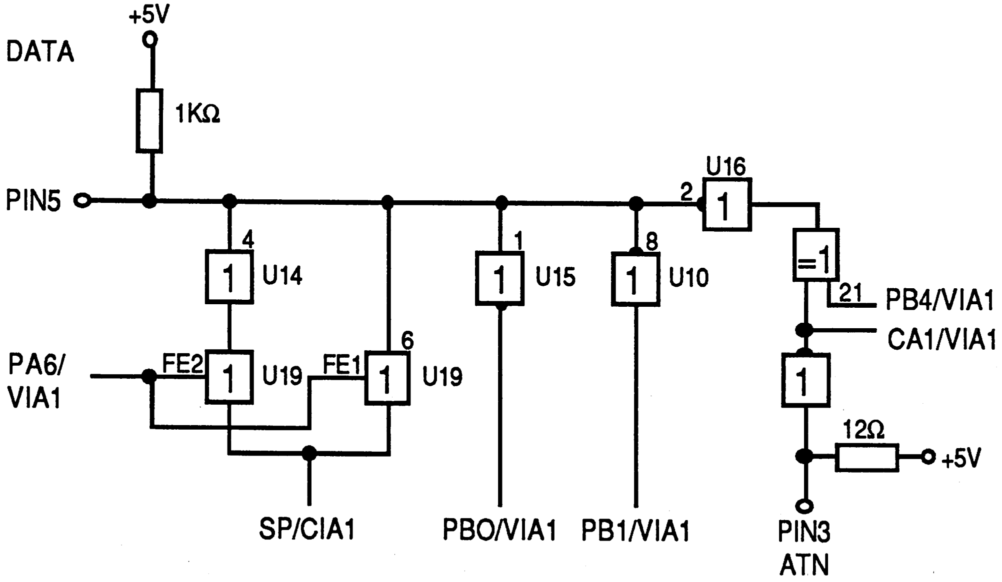
But as you might have already noticed, just 3500 bytes are loaded in the fast mode. This is the fault of the management routines in the disk drive and computer, neither of which is very well-written and therefore slow down the transfer.

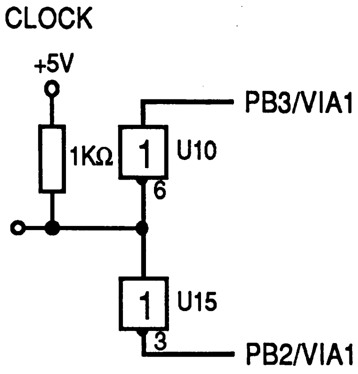
This was also the case with the C-64. The transfer algorithm of the C-64 bus is capable of sending or receiving at up to 1200 bytes per second. The management routines of the operating system slow the bus speed down to a meager 400 bytes per second.

This is why saving programs on the C-128 is no faster than on a C-64, although the fast bus is used. It is irrelevant whether or not data can be transferred in a millisecond or a few microseconds if the operating system can accept only one byte every 2.5 milliseconds.

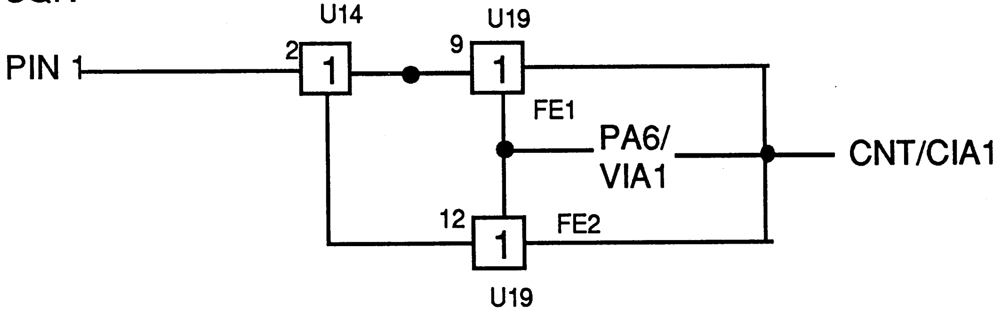
The fast-load systems are faster only because the management time (open file, manage pointers, ...) is drastically reduced.

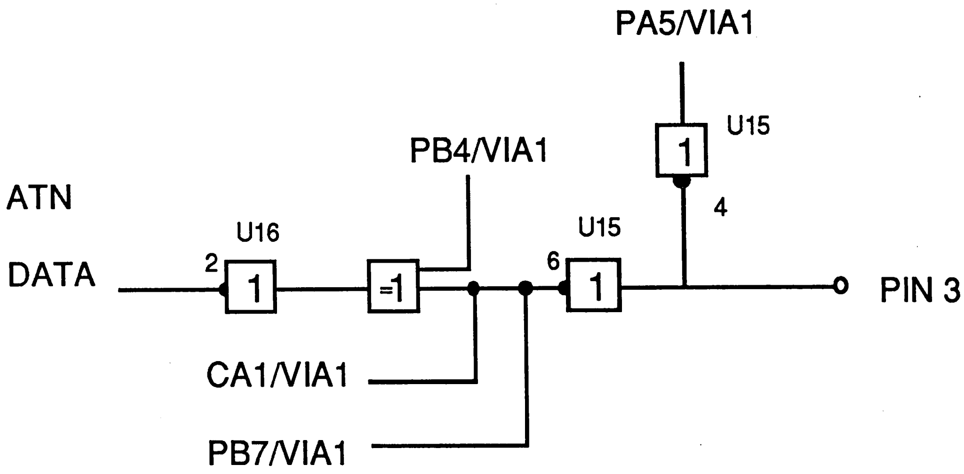
Theoretically it is even possible to realize bus transfers at up to 60,000 bytes per second with the C-128's fast bus hardware.


DATA
+5V
1KΩ
PIN5
U16
2
1
4
1
U14
1
1
U15
8
1
U10
=1
21
PB4/VIA1
CA1/VIA1
PA6/
VIA1
FE2
1
U19
FE1
6
1
U19
1
12Ω
+5V
SP/CIA1
PBO/VIA1
PB1/VIA1
PIN3
ATN


CLOCK
+5V
PB3/VIA1
1KΩ
1
U10
6
1
U15
3
PB2/VIA1

SQR


PIN 1
U14
2
1
U19
9
1
FE1
PA6/
VIA1
CNT/CIA1
12
1
FE2
U19


ATN
DATA
U16
2
1
=1
PB4/VIA1
CA1/VIA1
PB7/VIA1
U15
6
1
PA5/VIA1
U15
1
4
PIN 3

5.2.7 The stepper motor

The stepper motor is a special device. A normal motor starts to run as soon as it is supplied with current. An example of a motor of this type is the drive motor which rotates the diskette.

This drive motor has a tacho-generator on its axle, a small component which determines the rotation count of the motor. The current supply of the drive motor is controlled with this measurement so that the motor runs at a constant speed.

The stepper, on the other hand, is a special motor which is controlled not by a steady current but by pulses. The motor moves an exact amount each time it receives a pulse. For the stepper motor in the C-1570/71 this is exactly 1.8 degrees. This means that after exactly 200 pulses the motor has completely exactly one revolution.

The rotation angle of a stepper motor is called a step, which makes the nomenclature more understandable. Each pulse moves the motor one step. The advantage of a stepper motor is that it can be rotated both forwards and backwards. The rotation direction is controlled by the signals STP0 and STP1 on the 1570/1571. These two bits must be seen as a 2-bit value. If one increases the value, the motor moves so that the head travels toward the outside, while if this value is decreased, the head moves to the inside.

On the 1570/1571 the head is moved one track by two steps. These step pulses may not follow each other too closely, of course, since the stepper motor needs a certain time in which to make each step. As you have certainly noticed, the head moves much faster in the 1571 than it does in the 1541 mode.

Since the clock frequency is twice as high in the 1571 mode, the step pulses are also created twice as fast. When experimenting with the stepper programming, you should proceed very carefully, and preferably use the operating system routines.

CHAPTER 6

THE DISK OPERATING SYSTEM (DOS)

6.1 The DOS routines

6.1.1 The DOS - An introduction

First a little history. The grandfather of 1571 DOS was the operating system of the CBM 4040 double disk drive. The CBM 4040 had a microcomputer controller with two processors. One processor was responsible for managing data and the other for controlling the drives. This division of labor was intended to make the system faster.

When the VIC-20, Commodore's first home computer, appeared on the market, it naturally had to have a compatible disk drive. It would have been costly to develop a completely new system—and it didn't make sense to reinvent the wheel when the capable CBM 4040 disk drive was available. A control circuit using one processor for a single disk drive was developed. The software of the 4040 drive was simply modified. The disk management routines were the same as those on the 4040. But the new single drive lacked the second processor for drive control. As a result, the processor of the VIC-1540 disk drive also had to take over the tasks of the control processor. This decreased the speed of the VIC-1540 drive.

The DOS in the VIC-1541 is almost identical to that in the VIC-1540. Only the bus routines were changed, since the C-64 has a slightly lower clock frequency than the VIC-20 which decreases the bus controller somewhat.

The 1571 DOS consists of 16K of ROM of the VIC-1541 and an additional 12K of new operating system components. Once again timeliness won out and Commodore simply expanded and adapted the existing DOS version again. The fact that the performance of the disk drive has not improved much is obvious.

Beyond this, the 1571 DOS V3.0 perfects chaos itself. This DOS contains a management section which was intended for two drives and a multi-processor system. This includes drive management that was intended for multi-processor operation but can control only one drive.

This section of the 1541 ROM was copied for use in the 1571 ROM in its entirety. Only a few routines were modified. New program sections, such as those to manage the second side of the diskette, were simply inserted. This also includes a new drive controller routine, called a job loop. In addition to the 1541 ROM, a whole set of new functions was implemented, which in particular handle the control of the WD 1770 controller.

All told, the 1570/71 DOS consists of a hodge-podge of program fragments simply grouped together. The sad part is that as a result the disk drive is not very powerful or efficient, although it offers many technical possibilities. The slow transfer and load rates are not the fault of any bus procedures, but are the product of the slow DOS management alone.

6.1.2 The most important DOS routines

The DOS consists of a myriad of different routines. Many of these you cannot use because they were intended as subroutines. Here is a short list of some interesting ROM routines:

```
$8162  Switch 1571 bus electronics to input.

$81CE  Switch 1571 bus electronics to output.

$85F9  Output byte on 1571 bus.
       The byte must be stored in $46.

$864B  Execute job.
       $F9 must contain the number of the buffer for the job
       is to be performed. The job table $00-$05 contains
       the job code. After execution of the job, X and A
       contain the return code of the job loop. If an error
       occurs during execution, the routine tries to execute
       the job again

$8764  Turn on drive motor.

$8770  Turn off drive motor.

$877C  Turn drive LED on.

$878B  Turn drive LED off.
```

$883C	Get error message from WD 1770. A and X contain the error code.
$884E	Pass command in A to the WD 1770 command register.
$8861	Wait for the WD 1770 to execute the command.
$89EF	Reset head to track 0.
$89FD	Test write protect A= $10: no write protect A= $00: write protect active
$9032	Switch to 1541 mode.
$904E	Switch to 1571 mode.
$93F3	Activate head on current disk side. C=0: side 1 C=1: side 2
$98D9	Convert 5 GCR bytes to 4 binary bytes.
$F6D0	Convert 4 binary bytes to 5 GCR bytes.
$FE00	Switch head to read.
$FE0E	Erase track (write with $55).

6.1.3 The zero-page

The zero-page is the memory area $0000 to $00FF, which can be accessed especially quickly by certain 6502 machine language instructions. For this reason, important parameters and data which the DOS requires are stored here.

The addresses $00 to $11 have a special significance. These memory locations are used to pass commands and parameters to the job loop, which controls the disk drive. A memory location which stores the command for the action to be performed is reserved for each buffer. The job loop returns a message in this memory location which tells whether the command was performed without error or not.

A large section of the zero-page is used for the management of files. Since the DOS is based on a two-drive system, a good deal of space is reserved for the second drive, which isn't even present.

Some zero-page locations contain constants important for the operation of the disk drive. These include:

```
57  $39  Marker for data block (8)
104 $68  Flag for the initialization method
106 $6A  Number of read attempts
```

These memory locations are initially set when the disk drive is turned on. After this the DOS always works with the values stored in these addresses. If, for example, you change the marker for a data block and then format a diskette in this manner, it cannot be read later with the normal marker. Another possibility is to define the number of read attempts as well as the behavior of the disk drive in the event of read errors in address $6A.

Appendix C contains the complete zero-page listing. So you don't have to hunt through the whole book to find it, we have placed it near the end of the book.

6.1.4 DOS V3.0 in detail

As we mentioned in Section 6.1.1, parts of the DOS are intended for double drive or multi-processor operation. But since it runs on a system with just one drive and one processor, the capabilities of the system are vastly underused. Furthermore, the management required for two drives is more extensive than that required for one, so the 1571 requires additional processing time without needing it.

The style of buffer management is an especially notable leftover from the 4040 double drives. One, two, or even three buffers are required per file depending on the type.

Since the disk drive can manage up to 5 files at a time, each file is assigned an internal channel. This channel is, in turn, assigned the required buffers. Beyond this there are tables which contain information about which buffers are currently needed, which data have not been processed yet, and so on.

As you see, an enormous amount of management work is necessary for even the smallest disk access, greatly reducing the speed of the 1571.

When working with the DOS, you might want to keep its history in mind. We emphasize that it was not developed in one pass, but arose from versions of the preceding disk drives modified for the new drive.

This DOS version has been changed, expanded, and extended three or four times. This increases the error rate, the amount of unnecessary management work, and above all, reduces the performance of the disk drive.

6.1.5 Errors in the DOS

Naturally, the development of an operating system is not without error. Errors have struck the 1570/1571 as well. This concerns some functions and commands which do not operate in the desired manner, such as the block-allocate command, the replace function, and so on.

In addition, there are some locations at which the ROM contains commands which make no sense. The largest group are the assembler instructions which do not make any sense, or are superfluous. The following addresses, among others, contain such constructions:

```
$85DA $9396 $9690 $A605 $E853 $E9DA $EAA7 $F258 $FF13
```

Some other DOS locations are erroneous. These errors are often so slight that they do not immediately make themselves known in a disk-drive system crash. Here is a short list of some mysterious DOS locations:

$8056: Here some flags are masked out of $37 which do not have any control function. This indicates that the wrong flags are being masked, whereby the instruction should read "AND $BE".

$8124/$826F: Here the flag for the real-time clock is activated (which is not even used in the DOS.) Since this action occurs in connection with the bus actions, it raises the suspicion that the nearby flag for the serial input/output register is intended.

$BF57/BF75: This is a jump to a location where no program exists.

$E69B: In this routine the SED command is used without disabling the interrupts. As a result, the job control loop will be called while the BCD arithmetic is activated. The fact that the proper control parameters will not be calculated should be obvious.

6.2 1570/1571 ROM listing

6.2.1 Listing comments

The ROM listing in Appendix A of this book differs from many other ROM listings in several respects. You may have noticed the curious superscripted (^{1}) numbers following some memory addresses, or you may have wondered what the numbers in square brackets ([ ]) mean. These involve a cross reference.

The specifications about each ROM routine, enclosed in square brackets, name all locations in ROM which call this routine. If another location in the routine jumped to, this entry address will be named, followed by a colon. The address of the calling point is then given. Here are some examples:

```
[1234/5678]       This ROM routine called from 1234 and 5678.
[EEEA:1652]       The address EEEA in the ROM routine is
                  called from 1652.
[5527:78ED,5652]  The address 5527 is called from 78ED and
                  5652.
```

In addition to these cross references, comments are often given when the location is called via a vector or a program routine. Also, the comment "`Routine not used`" occurs from time to time. Furthermore, comparison addresses are given for some routines, in the form "`cmp 1234`" for example. These indicate that the same routine or a routine which performs the same function occurs at a different location in ROM. You should follow these references with interest in any event. If two identical routines are present, they are usually not commented identically. This way you can work with both versions of comments and therefore have a better, more comprehensive explanation of the ROM routine.

Another type of cross reference is the superscripted numbers which appear after some addresses. These indicate that this address is called. This usually involves a relative jump instruction. This means that the locations from which these addresses are called appear 128 bytes before or after the address, which corresponds roughly to a page backward or forward. The number indicates how many such references exist. If you still cannot find an entry point, it may be that the data for this address are given in the header in the square brackets. Always check the header first.

Why are these cross references necessary? Take a look at address $93F3 in the ROM listing. A routine with a branch instruction is found at this address. It's necessary to determine what value the carry flag had when the routine was called. The specified addresses [895C, 9371 and 9B41] indicate the locations from which the routine $93F3 is called. From here you can determine what value the carry flag had.

In addition, these cross references indicate which routines are used often. You can also tell that half of all DOS routines are called only once. Basically this does not involve subroutines, but program sections. The superscripted cross references usually have the value 1. But these cases are not particularly interesting. You should direct your attention to locations which are called from more than one point. In this manner you can understand the flow of a routine more quickly.

In conclusion, a few words about the comments themselves. An attempt was made to comment all of the lines in the ROM listing. For some locations this was not very interesting, since it's hard to write exciting comments when the program itself is not exciting. At other locations a single line did not suffice to explain the routine. In these cases a small section with detailed explanations is often included.

6.3 1571 DOS Reader

The following short program allows you to read sections of the 1570/1571 DOS into the C-128 memory. You may then examine or modify the machine language routines using the C-128's built in machine language monitor. Input is done in hexadecimal and the contents of DOS memory are transferred to the same locations in C-128 memory. Some areas of the DOS will have to be transferred into different memory locations or banks in the C-128 so that memory conflicts do not occur. This is accomplished by changing the value of variable `A` in line 140 (`POKE (A-VALUE)`).

```
10  PRINT CHR$(147)"1571 ROM READER TO C-128
    MEMORY":PRINT
20  OPEN 1,8,15
30  INPUT "STARTING ADDRESS";A$
40  A = DEC(A$)
50  INPUT "ENDING ADDRESS  ";B$
60  B = DEC(B$)
70  HI = INT (A/256)
80  LO = A-256*HI
90  PRINT#1,"M-R";CHR$(LO);CHR$(HI)
100 REM READ DOS MEMORY
110 GET#1,A$
120 PRINT CHR$(19)CHR$(17)CHR$(17)CHR$(17)CHR$(17)
    "CURRENT ADDRESS    ";HEX$(A)
130 IF A = B THEN 170
140 POKE A,ASC(A$)
    :REM BANK1:POKE A,ASC(A$):BANK 15: REM BANK 1
150 A = A+1
160 GOTO 80
170 MONITOR
```

Appendices

Appendix A

1571 DOS Listing

(ROM Version 03)

```
---------------------------------------------------------------------
8000 92 25                        ROM checksum [used: 929D/92A4]
---------------------------------------------------------------------
Author acknowledgement
8002 53 2F 57 20 2D 20 44 41      S/W - DAVID G
800A 56 49 44 20 47 20 53 49      SIRACUSA
8012 52 41 43 55 53 41 0D 48      H/W - GREG
801A 2F 57 20 2D 20 47 52 45      BERLIN
8022 47 20 42 45 52 4C 49 4E      1985
802A 0D 31 39 38 35 0D

------------------------------------------------------------------
[CB63/806D:Vectors 80BE,80C0,80C6,80C8]
Routine for User-0-command('U0')
8030   AD 74 02   LDA $0274     Get length of command string and
8033   C9 03      CMP #$03      test against smallest cmnd length
8035   90 2E      BCC $8065     Is the command less than 3 chars?
8037   AD 02 02   LDA $0202     NO, then get and note
803A   85 3B      STA $3B       command number
803C   29 1F      AND #$1F      Limit number to range of 0-31 and
803E   AA         TAX           mask control flag
803F   0A         ASL A         Double value (2-byte pointer in
8040   A8         TAY           table) and set as pointer
8041   B9 8E 80   LDA $808E,Y   Get and set low-byte of above
8044   85 75      STA $75       routine
8046   B9 8F 80   LDA $808F,Y   Take up high-byte of address
8049   85 76      STA $76       in pointers $75/$76
804B   E0 1E      CPX #$1E      check against 1541 status command
804D   F0 07      BEQ $8056     Should a new command be executed?
804F   AD 0F 18   LDA $180F     NO-Get flag for 1571/1541 range
8052   29 20      AND #$20      and test it
8054   F0 0F      BEQ $8065     Is drive in 1571 mode?
8056¹  A5 37      LDA $37       YES— [Error; see 7.1.5]
8058   29 EB      AND #$EB      [useless bitflags will be]
805A   85 37      STA $37       [masked out]
805C   BD 6E 80   LDA $806E,X   Set jobcode of equivalent
805F   8D 02 02   STA $0202     disk controller command
8062   6C 75 00   JMP ($0075)   Call routine
8065¹  A9 EA      LDA #$EA      Set pointer for
8067   85 6B      STA $6B       table of 1541
8069   A9 FF      LDA #$FF      user command
806B   85 6C      STA $6C       to $FFEA
806D⁴  60         RTS           Return from subroutine
```

```
[805C]   Jobcodes to command routines
806E 80 81 90 91 B0 B1 F0 F1      Bit0 = Drive number
8076 00 01 B0 01 00 01 00 01      Bit1-7 : $80 = Read / $90 = Write
807E 80 81 90 91 B0 B1 F0 F1      $B0 = Look for sector/$F0=Format
8086 00 01 B0 01 00 01 00 80      $00 = No job (Other function)
------------------------------------------------------------------
[8041]  Addresses of command routines w/  'User0'
        Command number:
        Bit0    : Drive (0/1)
        Bit1-3  : function
        Bit4    : Diskette side  (0/1)
808E 71 83        0 / 00         $8371   Read CP/M sector
8090 7F 83        1 / 01         $837F   Error:'Drive Not Ready'
8092 EC 83        2 / 02         $83EC   Write CP/M sector
8094 F8 83        3 / 03         $83F8   Error:'Drive Not Ready'
8096 8B 84        4 / 04         $848B   Read CP/M sectorheader
8098 7F 83        5 / 05         $837F   Error:'Drive Not Ready'
809A B7 84        6 / 06         $84B7   Format CP/M diskette
809C B7 84        7 / 07         $84B7   Format CP/M diskette
809E F1 84        8 / 08         $84F1   Get/set CP/M sector set-up
80A0 F1 84        9 / 09         $84F1   Get/set CP/M sector set-up
80A2 17 85       10 / 0A         $8517   Determine CP/M sector seq.
80A4 7F 83       11 / 0B         $837F   Error:'Drive Not Ready'
80A6 6B 85       12 / 0C         $856B   Get/set cmmand-status byte
80A8 7F 83       13 / 0D         $837F   Error:'Drive Not Ready'
80AA A5 85       14 / 0E         $85A5   Display 'Syntax Error(31)'
80AC A5 85       15 / 0F         $85A5   Display 'Syntax Error(31)'
80AE 71 83       16 / 10         $8371   Read CP/M sector
80B0 7F 83       17 / 11         $837F   Error:'Drive Not Ready'
80B2 EC 83       18 / 12         $83EC   Write CP/M sector
80B4 F8 83       19 / 13         $83F8   Error:'Drive Not Ready'
80B6 8B 84       20 / 14         $848B   Read CP/M sector header
80B8 7F 83       21 / 15         $837F   Error:'Drive Not Ready'
80BA B7 84       22 / 16         $84B7   Format CP/M diskette
80BC B7 84       23 / 17         $84B7   Format CP/M diskette
80BE 6D 80       24 / 18         $806D   No function (rts)
80C0 6D 80       25 / 19         $806D   No function (rts)
80C2 17 85       26 / 1A         $8517   Determine sector sequence
80C4 7F 83       27 / 1B         $837F   Error:'Drive Not Ready'
80C6 6D 80       28 / 1C         $806D   No function (rts)
80C8 6D 80       29 / 1D         $806D   No function (rts)
80CA E5 8F       30 / 1E         $8FE5   Execute 1571 status comand
80CC 80 90       31 / 1F         $9080   Load file over 1571 bus
```

```
[A7BA]
Take command from serial bus (ATN encountered)
80CE   78          SEI           Disable bus/controller interrupt
80CF   A9 00       LDA #$00      Clear pointer and flags :
80D1   85 7C       STA $7C       Receive flag for ATN from bus
80D3   85 79       STA $79       Flag for Listen
80D5   85 7A       STA $7A       Flag for Talk
80D7   A2 45       LDX #$45      Stack pointer
80D9   9A          TXS           initialization
80DA   20 B2 81    JSR $81B2     Switch 1571 bus for input
80DD   A9 80       LDA #$80      Set flag for last char (EOI = End
80DF   85 F8       STA $F8       of Information)
80E1   85 7D       STA $7D       Clear flag for 'ATN observed'
80E3   20 B7 E9    JSR $E9B7     Clock output to low
80E6   20 A5 E9    JSR $E9A5     Data output to high
80E9   AD 00 18    LDA $1800     Get bus control register and
80EC   09 10       ORA #$10      set ATN output
80EE   8D 00 18    STA $1800     to high
80F1[1] AD 00 18   LDA $1800     Check ATN input
80F4   10 64       BPL $815A     Is ATN set?
80F6   29 04       AND #$04      YES— Get clock input
80F8   D0 F7       BNE $80F1     Is clock set?
80FA[1] 20 CA 82   JSR $82CA     NO—Get command byte from bus
80FD   C9 3F       CMP #$3F      Compare with value for 'Unlisten'
80FF   D0 0C       BNE $810D     Should Listener complete work?
8101   A5 37       LDA $37       YES—Get bus control flag
8103   29 BF       AND #$BF      and set flag for
8105   85 37       STA $37       '1541-bus mode'
8107   A9 00       LDA #$00      Clear flag for
8109   85 79       STA $79       Listen
810B   F0 0E       BEQ $811B     Jump to $811B
810D[1] C9 5F      CMP #$5F      Compare with value for 'Untalk'
810F   D0 0D       BNE $811E     Should Talker finish its work?
8111   A5 37       LDA $37       YES—Get bus control flag
8113   29 BF       AND #$BF      and set Flag for
8115   85 37       STA $37       '1541-bus mode'
8117   A9 00       LDA #$00      Clear Flag for
8119   85 7A       STA $7A       Talk
811B[1] 4C 92 81   JMP $8192     Wait until ATN-mode is available
811E[1] C5 78      CMP $78       Compare w/device address for Talk
8120   D0 0A       BNE $812C     Is Talk addressed?
8122   A9 01       LDA #$01      YES—Set flag for
8124   85 7A       STA $7A       'Talk receive'
8126   A9 00       LDA #$00      Clear Flag for
8128   85 79       STA $79       Listen
812A   F0 29       BEQ $8155     Jump to $8155
```

```
812C1 C5 77       CMP $77        Compare w/ device addr for Listen
812E  D0 0A       BNE $813A      Is Listen addressed?
8130  A9 01       LDA #$01       YES—Set Flag for
8132  85 79       STA $79        'listen received'
8134  A9 00       LDA #$00       Clear flag for
8136  85 7A       STA $7A        Talk
8138  F0 1B       BEQ $8155      Jump to $8155
813A1 AA          TAX            Mark ATN-command
813B  29 60       AND #$60       Isolate cntrl bits/Talk & Listen
813D  C9 60       CMP #$60       & test against value f/'both set'
813F  D0 4C       BNE $818D      Will channel # be transmitted?
8141  8A          TXA            YES—Repeat and set
8142  85 84       STA $84        original secondary address;
8144  29 0F       AND #$0F       Establish and set number of
8146  85 83       STA $83        abovementioned disk channel;
8148  A5 84       LDA $84        Get orig/2ndary addr(ATN-command)
814A  29 F0       AND #$F0       and isolate command bits
814C  C9 E0       CMP #$E0       Is Bit7 (Open/Close) also set?
814E  D0 42       BNE $8192      Should the channel be closed?
8150  58          CLI            YES—Enable bus/controler interupt
8151  20 C0 DA    JSR $DAC0      Close channel&close current files
8154  78          SEI            Disable bus/controller interrupt
81552 2C 00 18    BIT $1800      Test ATN input
---------------------------------------------------------------------
[A9B3]  Steer bus to ATN command
8158  30 A0       BMI $80FA      Is ATN still set?
815A3 A9 00       LDA #$00       NO—Clear flag for
815C  85 7D       STA $7D        'ATN active'
815E  AD 00 18    LDA $1800      Get bus control register
8161  29 EF       AND #$EF       and clear ATN output
8163  8D 00 18    STA $1800      again
8166  A5 79       LDA $79        Get flag for Listen
8168  F0 0D       BEQ $8177      Is the bus in Listener mode?
816A  24 37       BIT $37        NO—Test bus control flag
816C  50 03       BVC $8171      Is bus in 1571 mode?
816E  20 99 81    JSR $8199      YES—Send DRF code
81711 20 42 83    JSR $8342      Take data from bus to
8174  4C 6B 83    JMP $836B      command waitloop
81771 A5 7A       LDA $7A        Get flag for talk
8179  F0 0F       BEQ $818A      Is the bus in Talker mode?
817B  20 9C E9    JSR $E99C      Set Data output to low
817E  20 AE E9    JSR $E9AE      Set Clock output to high
8181  20 83 A4    JSR $A483      approx. 80 cycle delay
8184  20 EA 81    JSR $81EA      Give data over bus after talk
8187  20 83 A4    JSR $A483      approx. 80 cycle delay
818A1 4C 66 83    JMP $8366      to command waitloop
818D1 A9 10       LDA #$10       Set ATN output to high
```

```
818F   8D 00 18    STA $1800       Set Data and Clock to low
8192³  2C 00 18    BIT $1800       Test ATN
8195   10 C3       BPL $815A       Is ATN set?
8197   30 F9       BMI $8192       YES—Wait til ATN is cleared again
-----------------------------------------------------------------
[816E/81A1]
Send DRF (Device Request Fast) to computer (fast bus mode)
8199   20 59 EA    JSR $EA59       Check for ATN command mode
819C   20 C0 E9    JSR $E9C0       Read bus reg. by constant values
819F   29 04       AND #$04        and isolate Clock input
81A1   D0 F6       BNE $8199       Is Clock set?
81A3   20 CE 81    JSR $81CE       NO—Switch 1571 to output
81A6   A9 00       LDA #$00        DRF Signal
81A8   8D 0C 40    STA $400C       in serial output register
81AB   A9 08       LDA #$08        Bitflag for serial register empty
81AD¹  2C 0D 40    BIT $400D       Get output status
81B0   F0 FB       BEQ $81AD       Is data byte transferred?
-----------------------------------------------------------------
[80DA/836B/846D/8591/8EAC/A61F/A7AD]
Switch 1571 bus to input
81B2   08          PHP             Retain processor status
81B3   78          SEI             Disable bus/controller interrupt
81B4   AD 0E 40    LDA $400E       YES—Get control register
81B7   29 BF       AND #$BF        Switch serial connection
81B9   8D 0E 40    STA $400E       to input
81BC   AD 0F 18    LDA $180F       Set control bit for
81BF   29 FD       AND #$FD        1571 bus turn to
81C1   8D 0F 18    STA $180F       input mode
81C4   A9 84       LDA #$84        [Error; see 7.1.5]
81C6   8D 0D 40    STA $400D       [Real-time clock not used]
81C9   2C 0D 40    BIT $400D       Reset last interrupt flag
81CC   28          PLP             Reset processor status
81CD   60          RTS             Return from this subroutine
-----------------------------------------------------------------
[81A3/8394/8461/84F6/8533/8582/8E93/8E9A/9080]
Switch 1571 bus to output
81CE   08          PHP             Retain processor status
81CF   78          SEI             Disable bus/controller interrupt
81D0   AD 0F 18    LDA $180F       Set control bit for
81D3   09 02       ORA #$02        1571 bus direction to
81D5   8D 0F 18    STA $180F       output mode
81D8   AD 0E 40    LDA $400E       Switch serial register
81DB   09 40       ORA #$40        to
81DD   8D 0E 40    STA $400E       output
81E0   A9 08       LDA #$08        Limit interrupt from
81E2   8D 0D 40    STA $400D       'byte input/output'
81E5   2C 0D 40    BIT $400D       Clear flag from last interrupt
```

```
81E8   28           PLP              Set processor status again
81E9   60           RTS              Return from this subroutine
-----------------------------------------------------------------------
[8184]  cf. E909
Output file data over bus (1571 mode)
81EA   78           SEI              Disable bus/controller interrupt
81EB   20 EB D0     JSR $D0EB        Determine internal channel number
81EE   B0 06        BCS $81F6        Has channel been found?
81F01  A6 82        LDX $82          YES—Get number of channel
81F2   B5 F2        LDA $F2,X        Determine appropriate bus status
81F4   30 01        BMI $81F7        Channel active?
81F61  60           RTS              NO—Return from this subroutine
81F71  20 59 EA     JSR $EA59        Test for ATN command mode
81FA   20 C0 E9     JSR $E9C0        Get constant value/bus cntrl reg
81FD   29 01        AND #$01         and test for Data input
81FF   08           PHP              Mark result
8200   20 B7 E9     JSR $E9B7        Set Clock output for high
8203   28           PLP              Repeat status of Data line
8204   F0 12        BEQ $8218        Is Data set?
82061  20 59 EA     JSR $EA59        YES—Test for ATN command mode
8209   20 C0 E9     JSR $E9C0        Get value--bus control registers
820C   29 01        AND #$01         and get status of data input
820E   D0 F6        BNE $8206        Is data still set?
8210   A6 82        LDX $82          NO—Get number of current channel
8212   B5 F2        LDA $F2,X        and get corresponding bus status
8214   29 08        AND #$08         Test EOI (End of Information)flag
8216   D0 14        BNE $822C        Is last byte being transferred?
82182  20 59 EA     JSR $EA59        NO—Check with ATN command mode
821B   20 C0 E9     JSR $E9C0        Get value of bus control register
821E   29 01        AND #$01         & determine status of Data input
8220   D0 F6        BNE $8218        Is Data set?
82221  20 59 EA     JSR $EA59        NO—Check against ATN command mode
8225   20 C0 E9     JSR $E9C0        Get val from bus control register
8228   29 01        AND #$01         and test status of Data input
822A   F0 F6        BEQ $8222        Is Data set now?
822C2  20 AE E9     JSR $E9AE        YES—Set Clock output to high
822F   20 59 EA     JSR $EA59        Test for ATN command mode
8232   20 C0 E9     JSR $E9C0        Get val from bus control register
8235   29 01        AND #$01         and isolate Data input
8237   D0 F3        BNE $822C        Is Data set?
8239   24 37        BIT $37          NO—Test flag for bus mode
823B   50 39        BVC $8276        Is Bus in 1571 mode?
-----------------------------------------------------------------------
```

```
Output byte over 1571 bus
823D  AD 0F 18   LDA $180F     YES—1571 bus circuitry
8240  09 02      ORA #$02      switched to
8242  8D 0F 18   STA $180F     output mode (Bit 1 = 1)
8245  AD 0E 40   LDA $400E     Turn serial
8248  09 40      ORA #$40      output register
824A  8D 0E 40   STA $400E     to output
824D  2C 0D 40   BIT $400D     Reset interrupt register
8250  A6 82      LDX $82       Number of current channel
8252  BD 3E 02   LDA $023E,X   Get data byte/channel to transfer
8255  8D 0C 40   STA $400C     and get status of output
8258¹ AD 0D 40   LDA $400D     register;
825B  29 08      AND #$08      See if output register is empty
825D  F0 F9      BEQ $8258     Is byte transferred?
825F  AD 0E 40   LDA $400E     YES—Switch serial register
8262  29 BF      AND #$BF      to an
8264  8D 0E 40   STA $400E     input register
8267  AD 0F 18   LDA $180F     Bus circuitry back
826A  29 FD      AND #$FD      to input
826C  8D 0F 18   STA $180F     mode (Bit1 =0)
826F  A9 84      LDA #$84      [Error—see 7.1.54]
8271  8D 0D 40   STA $400D     [Real-time clock not used here]
8274  D0 3C      BNE $82B2     Jump to $82B2
------------------------------------------------------------------
Output byte over 1541 bus
8276¹ A9 08      LDA #$08      Set number of bits per byte
8278  85 98      STA $98       in counter
827A¹ 20 C0 E9   JSR $E9C0     Get bus control register and
827D  29 01      AND #$01      check Data input
827F  D0 43      BNE $82C4     Is Data set?
8281¹ A6 82      LDX $82       NO—Get current channel number and
8283  BD 3E 02   LDA $023E,X   determine appropriate data byte
8286  6A         ROR A         Take a bit from there & mark the
8287  9D 3E 02   STA $023E,X   remainder of the byte
828A  B0 05      BCS $8291     Is Bit at 1?
828C  20 A5 E9   JSR $E9A5     NO—Set Data output to high
828F  D0 03      BNE $8294     Jump to $8294
8291¹ 20 9C E9   JSR $E99C     Switch Data output to low
8294¹ 20 7E A4   JSR $A47E     approx. 45 cycle delay
8297  A5 23      LDA $23       Flag for 1541/1540 bus delay
8299  D0 E6      BNE $8281     Is bus in 1541 mode?
829B  20 83 A4   JSR $A483     YES—approx. 80 cycles delay
829E  20 B7 E9   JSR $E9B7     Set Clock output to low
82A1  20 7E A4   JSR $A47E     approx. 45 cycle delay
82A4  A5 23      LDA $23       Flag for 1541/1540 bus delay
82A6  D0 03      BNE $82AB     Is bus in 1541 mode?
82A8  20 83 A4   JSR $A483     YES—approx. 80 cycle delay
```

```
82AB¹ 20 FB FE   JSR $FEFB    Set Clock on high and Data on low
82AE  C6 98      DEC $98      Counter for bits transferred
82B0  D0 C8      BNE $827A    Is byte transferred?
82B2² 20 59 EA   JSR $EA59    YES—Test for ATN command mode
82B5  20 C0 E9   JSR $E9C0    Get bus control register and
82B8  29 01      AND #$01     take up DATA input
82BA  F0 F6      BEQ $82B2    Is Data set?
82BC  58         CLI          YES—Enable bus/controler interupt
82BD  20 AA D3   JSR $D3AA    Read next byte from file
82C0  78         SEI          Disable bus/controller interrupt
82C1  4C F0 81   JMP $81F0    get ready for output again
82C4¹ 4C 62 83   JMP $8362    Back to command wait loop
-----------------------------------------------------------------
[8358]
Read command byte from 1571 bus
82C7  2C 0D 40   BIT $400D    Reset interrupt register
82CA  A9 08      LDA #$08     Determine # of bits to transfer
82CC  85 98      STA $98      per byte
82CE¹ 20 59 EA   JSR $EA59    Test for ATN command mode and
82D1  20 C0 E9   JSR $E9C0    read bus control register
82D4  29 04      AND #$04     Test Clock input
82D6  D0 F6      BNE $82CE    Is Clock set?
82D8  20 9C E9   JSR $E99C    NO—Set Data output to high
82DB  A9 01      LDA #$01     Test Data input in
82DD¹ 2C 00 18   BIT $1800    bus control register
82E0  D0 FB      BNE $82DD    Is Data still set?
82E2  8D 05 18   STA $1805    Set Timer 1 (highbyte)    (1)
82E5¹ 20 59 EA   JSR $EA59    Test for ATN command mode
82E8  AD 0D 18   LDA $180D    Test interrupt flag for
82EB  29 40      AND #$40     'Timer 1 running'
82ED  D0 09      BNE $82F8    Have 256 time-cycles passed?
82EF  20 C0 E9   JSR $E9C0    NO-Get val f/bus control register
82F2  29 04      AND #$04     and test Clock input
82F4  F0 EF      BEQ $82E5    Is Clock set?
82F6  D0 19      BNE $8311    YES—Jump to $8311
82F8¹ 20 A5 E9   JSR $E9A5    Set Data output to high
82FB  A2 18      LDX #$18     Wait loop:
82FD¹ CA         DEX          Wait about
82FE  D0 FD      BNE $82FD    0.1 ms
8300  20 9C E9   JSR $E99C    Set Data output to low
8303¹ 20 59 EA   JSR $EA59    Test for ATN command mode
8306  20 C0 E9   JSR $E9C0    Get value of bus control register
8309  29 04      AND #$04     and isolate Clock input
830B  F0 F6      BEQ $8303    Is Clock still set?
830D  A9 00      LDA #$00     YES—Set flag:'last byte received'
830F  85 F8      STA $F8      (EOI)
8311³ AD 00 18   LDA $1800    Determine, invert and mark
```

```
8314   49 01      EOR #$01         value of
8316   AA         TAX              data input
8317   AD 0D 40   LDA $400D        Get flag:'serial input register
831A   29 08      AND #$08         full'
831C   F0 08      BEQ $8326        Has a byte been received?
831E   AD 0C 40   LDA $400C        YES—Read byte out of register
8321   85 85      STA $85          and save current data byte;
8323   4C 3C 83   JMP $833C        end
8326¹  8A         TXA              Get inverted data value again
8327   4A         LSR A            and save in Carry
8328   29 02      AND #$02         Test Clock input
832A   D0 E5      BNE $8311        Was Clock set simultaneously?
832C   66 85      ROR $85          YES—Take data bit in data byte
832E¹  20 59 EA   JSR $EA59        Test on ATN-command mode
8331   20 C0 E9   JSR $E9C0        Get bus control register
8334   29 04      AND #$04         Test Clock input
8336   F0 F6      BEQ $832E        Is Clock still set?
8338   C6 98      DEC $98          NO—Counter for # of data bits
833A   D0 D5      BNE $8311        Is entire byte received?
833C¹  20 A5 E9   JSR $E9A5        YES—Set Data output to low
833F   A5 85      LDA $85          Get Data byte
8341   60         RTS              Return from this subroutine
-----------------------------------------------------------------
[8171/835F]  cf EA2E
Take byte from bus
8342   78         SEI              Disable bus/controller interrupt
8343   20 07 D1   JSR $D107        Determine internal channel number
8346   B0 05      BCS $834D        Has channel been found?
8348   B5 F2      LDA $F2,X        YES—Status of channel
834A   6A         ROR A            Test flag for write mode
834B   B0 0B      BCS $8358        Is channel opened for writing?
834D¹  A5 84      LDA $84          NO—Get current secondary address
834F   29 F0      AND #$F0         and command bits; test against
8351   C9 F0      CMP #$F0         'close channel '
8353   F0 03      BEQ $8358        Should channel be ended?
8355   4C 66 83   JMP $8366        NO—Return to command waitloop
8358²  20 C7 82   JSR $82C7        Get byte from 1571 bus
835B   58         CLI              Enable bus/controller interrupt
835C   20 B7 CF   JSR $CFB7        Write byte in file
835F   4C 42 83   JMP $8342        some more
-----------------------------------------------------------------
[82C4]
Set bus back; return to command waitloop
8362   A9 00      LDA #$00         Clear bus
8364   85 37      STA $37          status byte
-----------------------------------------------------------------
```

```
[818A/8355]  Set bus back; maintain mode
8366  A9 00     LDA #$00        Set Data- and Clock output
8368  8D 00 18  STA $1800       to low
----------------------------------------------------------------
[8174/E698/E8EA/EA53]
Wait for next command
836B  20 B2 81  JSR $81B2       Switch 1571 bus to input mode
836E  4C E7 EB  JMP $EBE7       Wait for next computer command
----------------------------------------------------------------
[Origin over vector in 808E/80AE throuh routine $8030]
Read CP/M sector; previous error test
8371  8D 4D 02  STA $024D       Save jobcode of routine
8374  85 5F     STA $5F         from table $806E
8376  AD 0D 18  LDA $180D       Test CA2 input(circuitry shows
8379  4A        LSR A           'Write Protect' has interrupted)
837A  90 18     BCC $8394       Has diskette been changed?
837C  A2 0B     LDX #$0B        YES—Error #:'ID Mismatch Error'
837E  2C        .byte $2C       Transfr next 2 bytes(bit command)
----------------------------------------------------------------
[Origin of vector; : 8090/8098/80A4/80A8/80B0/80B8/80C4 thru $8030]
Display error 'drive not ready'
837F  A2 4F     LDX #$4F        Error # for 'drive not ready'
----------------------------------------------------------------
[83C7/844A/84B4/8E42/8384:8DBC]
Combine command status flag and output with error
8381  20 E9 85  JSR $85E9       Set up byte for output
8384  20 81 85  JSR $8581       Output message over 1571 bus
----------------------------------------------------------------
[84EE]  eventual error output (else return)
8387  E0 02     CPX #$02        Compare # with value for 'OK'
8389  B0 01     BCS $838C       Is an error set?
838B  60        RTS             NO—Return from this subroutine
----------------------------------------------------------------
[8389/8484/8568/875C]
Output error message (number in X)
838C  8A        TXA             Get error number and
838D  29 0F     AND #$0F        determine proper error number
838F  A2 00     LDX #$00        Set buffer number 0
8391  4C 0A E6  JMP $E60A       Prepare message text
----------------------------------------------------------------
[837A]
Read CP/M sector
8394  20 CE 81  JSR $81CE       Switch 1571 bus for output
8397  24 5E     BIT $5E         Get command status byte
8399  10 05     BPL $83A0       Is flag set for IBM-34 diskette?
839B  A9 09     LDA #$09        YES—Execute routine at $8D67
839D  4C E6 86  JMP $86E6       (read IBM system-34 sector)
```

```
83A0¹ 20 3D C6   JSR $C63D       Initialize Commodore diskette
83A3¹ 58         CLI             Enable bus/controller interrupt
83A4  A5 3B      LDA $3B         Get command number and test
83A6  29 20      AND #$20        'sector not read' flag
83A8  D0 26      BNE $83D0       Only buffer to be transferred?
83AA  AD 03 02   LDA $0203       NO—Get fourth char. from command
83AD  85 06      STA $06         string; take up as track number
83AF  AD 04 02   LDA $0204       Get fifth char. and set as sector
83B2  85 07      STA $07         number of job
83B4  A2 00      LDX #$00        Choose buffer number 0
83B6  A5 5F      LDA $5F         Get current jobcode and
83B8  95 00      STA $00,X       give to job loop
83BA  20 5E 86   JSR $865E       Execute job
83BD  78         SEI             Disable bus/controller interrupt
83BE  20 E9 85   JSR $85E9       Prepare return message for output
83C1  24 3B      BIT $3B         Test flag for 'error test'
83C3  70 04      BVS $83C9       Return message to be considered?
83C5  E0 02      CPX #$02        YES-Test return jobmessage w/'OK'
83C7  B0 B8      BCS $8381       Job run error-free?
83C9¹ 20 F9 85   JSR $85F9       YES-Send return message over bus
83CC  A5 3B      LDA $3B         Test flag for 'output buffer'
83CE  30 0D      BMI $83DD       Buffer transferred to computer
83D0¹ A0 00      LDY #$00        YES-Buffer pntr to start of buffr
83D2¹ B9 00 03   LDA $0300,Y     Get byte from buffer and set as
83D5  85 46      STA $46         character to be given
83D7  20 F9 85   JSR $85F9       Output character over 1571 bus
83DA  C8         INY             Turn buffer pointer to next byte
83DB  D0 F5      BNE $83D2       Entire buffer been transferred?
83DD¹ CE 05 02   DEC $0205       YES—Number of sector to be read
83E0  F0 06      BEQ $83E8       All sectors already?
83E2  20 1E 86   JSR $861E       NO—Set number of next sector
83E5  4C A3 83   JMP $83A3       Read next sector
83E8¹ 58         CLI             Enable bus/controller interrupt
83E9  4C AF 85   JMP $85AF       Get new track and set it
------------------------------------------------------------------
[Originates over vectors in 8092/80B2 through routine $8030]
Write CP/M sector; previous error check
83EC  8D 4D 02   STA $024D       Save jobcode
83EF  AD 0D 18   LDA $180D       Test CA2 input (Circuitry shows
83F2  4A         LSR A           'write protect' has interrupted)
83F3  90 0D      BCC $8402       Has diskette been exchanged?
83F5  A2 0B      LDX #$0B        YES-error #:'ID Mismatch Error'
83F7  2C         .byte $2C       Jump to next 2 bytes(bit command)
------------------------------------------------------------------
```

```
[Origin from vector in 8094 through routine $8030]
83F8  A2 4F     LDX #$4F        Error: 'drive not ready'
83FA  86 46     STX $46         set as character to be given
83FC  A5 3B     LDA $3B         Transfer 'error found'
83FE  09 08     ORA #$08        flag into
8400  85 3B     STA $3B         command number
--------------------------------------------------------------
[83F3]
Write CP/M sector
8402  24 5E     BIT $5E         Test command status byte
8404  10 05     BPL $840B       Flag for IBM-34 diskette set?
8406  A9 0A     LDA #$0A        YES—Execute routine at $8DF6
8408  4C E6 86  JMP $86E6       (Write IBM System 34 CP/M sector)
840B1 20 3D C6  JSR $C63D       Initialize Commodore diskette
840E  A5 3B     LDA $3B         Flag: 'Buffer read from computer'
8410  30 29     BMI $843B       Is flag in command byte set?
84121 78        SEI             YES—Disable bus/controler intrupt
8413  A0 00     LDY #$00        Set buffer pntr to start-of-buffr
84151 AD 00 18  LDA $1800       Get bus control register
8418  49 08     EOR #$08        Switch status of Clock output
841A  2C 0D 40  BIT $400D       Set interrupt register back
841D  8D 00 18  STA $1800       Set new Clock output value
84201 AD 00 18  LDA $1800       Test ATN input
8423  10 03     BPL $8428       Is ATN set?
8425  20 59 EA  JSR $EA59       YES—Test for ATN command mode
84281 AD 0D 40  LDA $400D       Test 'Byte in serial register
842B  29 08     AND #$08        received' flag
842D  F0 F1     BEQ $8420       Has a byte been read in?
842F  AD 0C 40  LDA $400C       YES—Get byte and write
8432  99 00 03  STA $0300,Y     in buffer 0
8435  C8        INY             Turn buffer pointer to next byte
8436  D0 DD     BNE $8415       Buffer already full
8438  20 B7 E9  JSR $E9B7       YES—Set Clock output to low
843B1 58        CLI             Enable bus/controller interrupt
843C  A5 3B     LDA $3B         Get command byte and test
843E  29 20     AND #$20        'Sector not written' flag
8440  D0 37     BNE $8479       Set?
8442  A5 3B     LDA $3B         NO—Retest 'error found'
8444  29 08     AND #$08        flag in command number
8446  F0 05     BEQ $844D       Should error be displayed?
8448  A6 46     LDX $46         YES—Get number of error
844A  4C 81 83  JMP $8381       and send over 1571 bus
844D1 AD 03 02  LDA $0203       Get 4h char. from command string
8450  85 06     STA $06         and set as track for job loop
8452  AD 04 02  LDA $0204       Get 5th char. from command string
8455  85 07     STA $07         and set as sector for job loop
8457  A2 00     LDX #$00        Choose buffer 0
```

```
8459    A9 90       LDA #$90       Give jobcode for 'write sector'
845B    95 00       STA $00,X      to job loop
845D    20 5E 86    JSR $865E      and execute
8460    78          SEI            Disable bus/controller interrupt
8461    20 CE 81    JSR $81CE      Switch 1571 bus to output
8464    20 E9 85    JSR $85E9      Prepare return message for output
8467    20 F9 85    JSR $85F9      Output byte over 1571 bus; wait
846A    20 A0 86    JSR $86A0      for shift from clock
846D    20 B2 81    JSR $81B2      Switch 1571 bus to input
8470    58          CLI            Enable bus/controller interrupt
8471    24 3B       BIT $3B        Test 'error test' flag
8473    70 04       BVS $8479      Should error return message be
8475    E0 02       CPX #$02       regarded? YES—Test error number
8477    B0 0B       BCS $8484      Is job running error-free?
8479[2] CE 05 02    DEC $0205      YES—Counter for sectors
847C    F0 09       BEQ $8487      Still a sector?
847E    20 1E 86    JSR $861E      YES—Calculate new sector number
8481    4C 12 84    JMP $8412      Run routine again
8484[1] 4C 8C 83    JMP $838C      Return to command waitloop
8488[1] 58          CLI            Enable bus/controller interrupt
8488    4C AF 85    JMP $85AF      Set new track and end
-------------------------------------------------------------------
[Origin over vectors 8096/80B6 of routine $8030]
Read next CP/M sector header(first System-34,then Commodore format)
848B    AD 02 02    LDA $0202      Get Jobcode
848E    29 01       AND #$01       and determine drive # from it
8490    D0 20       BNE $84B2      Is drive 0 contacted?
8492    A9 01       LDA #$01       YES—Clear 'Write protect has been
8494    8D 0D 18    STA $180D      interrupted' (disk exchange)
8497    A9 05       LDA #$05       Execute routine at $8A09
8499    20 E6 86    JSR $86E6      (Read IBM System 34)
849C    AE B0 01    LDX $01B0      Get return message and compare
849F    E0 02       CPX #$02       with value for 'Ok'
84A1    90 11       BCC $84B4      Run into an error?
84A3    A2 00       LDX #$00       YES—Clear command status byte
84A5    86 5E       STX $5E        Save the
84A7    A9 B0       LDA #$B0       jobcode for 'Search sector'
84A9    8D 4D 02    STA $024D      and give to
84AC    95 00       STA $00,X      job loop
84AE    20 5E 86    JSR $865E      Execute job
84B1    2C          .byte $2C      Jump to next 2 bytes(bit command)
84B2[1] A2 4F       LDX #$4F       Error # for 'drive not ready'
84B4[1] 4C 81 83    JMP $8381      Display return message,next cmd
-------------------------------------------------------------------
```

```
[Origin over vectors 809A/809C/80BA/80BC through routine $8030]
Format CP/M diskette
84B7   AD 02 02   LDA $0202         Get jobcode and
84BA   29 01      AND #$01          determine drive to be utilized
84BC   D0 2B      BNE $84E9         Should format be done in dr. 0?
84BE   AD 03 02   LDA $0203         YES—Get flag for diskette type
84C1   10 05      BPL $84C8         Is Commodore format desired?
84C3   A9 08      LDA #$08          NO—Format disk in IBM System34
84C5   4C E6 86   JMP $86E6         format (routine $8C57)
84C81  A9 00      LDA #$00          Clear command status byte
84CA   85 5E      STA $5E           (delete)
84CC   85 FF      STA $FF           Set drive status to 'ready'
84CE   AD 04 02   LDA $0204         Get 5th char. from command string
84D1   85 12      STA $12           and take on as first ID character
84D3   AD 05 02   LDA $0205         Get 6th char. from command string
84D6   85 13      STA $13           and store as 2nd character of ID
84D8   20 07 D3   JSR $D307         Close all channels
84DB   A9 01      LDA #$01          Set track number 1 as
84DD   85 80      STA $80           current track
84DF   A9 FF      LDA #$FF          Format disk in 1571/1541 format
84E1   8D 98 02   STA $0298         Get return message
84E4   20 89 A9   JSR $A989         and prepare for output
84E7   AA         TAX               Send message over 1571 bus, end
84E8   2C         .byte $2C         Jump to next 2 bytes(bit command)
84E91  A2 4F      LDX #$4F          Error # for 'drive not ready'
84EB   20 E9 85   JSR $85E9         Prepare byte for output
84EE   4C 87 83   JMP $8387         Prepare error message
-----------------------------------------------------------------
[Origin of vector 809C/80A0 through routine $8030]
Get /set CP/M sector format
84F1   78         SEI               Disable bus/controller interrupt
84F2   24 3B      BIT $3B           check command number
84F4   10 0A      BPL $8500         read sector format?
84F6   20 CE 81   JSR $81CE         YES—switch 1571 bus for output
84F9   A5 3C      LDA $3C           Get sector format and store
84FB   85 46      STA $46           as byte to be output
84FD   4C F9 85   JMP $85F9         Send byte over 1571 bus
85001  AE 74 02   LDX $0274         Determine length of comand string
8503   E0 04      CPX #$04          & test if exactly 3 char. long
8505   B0 0A      BCS $8511         Exactly 3 char. in buffer?
8507   A2 0E      LDX #$0E          YES—Error code for 'Syntax Error'
8509   20 E9 85   JSR $85E9         Prepare error for output
850C   A9 31      LDA #$31          Error message
850E   4C C8 C1   JMP $C1C8         Output '31 Syntax Error'
85111  AD 03 02   LDA $0203         Get byte f/cmd string(4th char)
8514   85 3C      STA $3C           and use as new sector format
8516   60         RTS               Return to caller
```

```
[Origin over vector in 80A2/80C2 by routine $8030]
Read CP/M sector header and determine sector sequence
8517   20 8B 84   JSR $848B      Read next header
851A   24 5E      BIT $5E        Test cmd status byte w/IBM flag
851C   10 48      BPL $8566      Is flag set for IBM-34 diskette?
851E   A9 0D      LDA #$0D       YES—Execute routine at $8F5F
8520   20 E6 86   JSR $86E6      (Set up sector sequence table)
8523   AE B0 01   LDX $01B0      Get return message; compare with
8526   E0 02      CPX #$02       value for 'Ok'
8528   B0 08      BCS $8532      Is there an error?
852A   20 61 89   JSR $8961      NO—Get lowest and highest sectors
852D   20 86 89   JSR $8986      Compute sector format
8530   8A         TXA            Get sector format
8531   48         PHA            and record it
8532¹  78         SEI            Disable bus/controller interrupt
8533   20 CE 81   JSR $81CE      Switch 1571 bus for output
8536   A5 5E      LDA $5E        Get command status byte,set as
8538   85 46      STA $46        character to be output
853A   20 F9 85   JSR $85F9      Send byte over 1571 bus
853D   AE B0 01   LDX $01B0      Get error number and compare
8540   E0 02      CPX #$02       with value for 'Ok'
8542   B0 23      BCS $8567      Is there an error?
8544   A5 97      LDA $97        NO—Set number of sectors in
8546   85 46      STA $46        track as character to be output
8548   20 F9 85   JSR $85F9      Send byte over 1571 bus
854B   A5 67      LDA $67        Set # of track read as character
854D   85 46      STA $46        to be output
854F   20 F9 85   JSR $85F9      Send byte over 1571 bus
8552   A5 60      LDA $60        Set smallest sector # as char.
8554   85 46      STA $46        to be output
8556   20 F9 85   JSR $85F9      Send byte over 1571 bus
8559   A5 61      LDA $61        Set greatest sector number as
855B   85 46      STA $46        character to be output
855D   20 F9 85   JSR $85F9      Send byte over 1571 bus
8560   68         PLA            Get sector format & set as char.
8561   85 46      STA $46        to be output
8563   20 F9 85   JSR $85F9      Send byte over 1571 bus
8566¹  60         RTS            Return to caller
8567¹  68         PLA            Fix stack
8568   4C 8C 83   JMP $838C      Display error message
-----------------------------------------------------------------
[Origin over vector in 80A6 by routine $8030]
Get command status byte / set it; get error number
856B   24 3B      BIT $3B        Test command number
856D   10 27      BPL $8596      Get command status byte?
856F   24 3B      BIT $3B        Test flag in command number
8571   50 0E      BVC $8581      Check for diskette exchange?
```

```
8573  AD 0D 18   LDA $180D      YES—Test hardware signal for
8576  4A         LSR A          'Write protect has interrupted'
8577  90 08      BCC $8581      Has diskette been exchanged?
8579  A5 5E      LDA $5E        Get command status byte
857B  29 F0      AND #$F0       and set up flag
857D  09 0B      ORA #$0B       Set error # for 'ID Mismatch'
857F  85 5E      STA $5E        and save as status byte
---------------------------------------------------------------
[8384/8571/8577]
Display command status byte
8581  78         SEI            Disable bus & controllr interrupt
8582  20 CE 81   JSR $81CE      Switch 1571 bus for output
8585  A5 5E      LDA $5E        Set status byte as character to
8587  85 46      STA $46        be sent
8589  20 F9 85   JSR $85F9      Send byte over 1571 bus
858C  A9 00      LDA #$00       Clear error flag (blink counter)
858E  8D 6C 02   STA $026C
8591  20 B2 81   JSR $81B2      Switch 1571 bus to input
8594  58         CLI            Enable bus/controller interrupt
8595  60         RTS            Return to caller
---------------------------------------------------------------
Set command status byte
8596¹ AD 03 02   LDA $0203      Get 4th char from command string,
8599  85 5E      STA $5E        and save as command status
859B  24 3B      BIT $3B        Test flag from command number
859D  50 05      BVC $85A4      diskette exchange be observed?
859F  A9 01      LDA #$01       YES—Initialize disk exchange flag
85A1  8D 0D 18   STA $180D      (write protect will interrupt)
85A4¹ 60         RTS            Return to caller
---------------------------------------------------------------
[Origin over vector in 80AA/80AC by routine $8030]
Display 'Syntax Error'
85A5  A2 0E      LDX #$0E       Set error number
85A7  20 E9 85   JSR $85E9      Prepare byte for output
85AA  A9 31      LDA #$31       Display
85AC  4C C8 C1   JMP $C1C8      '31 Syntax Error'  message
---------------------------------------------------------------
[83E9/8488]
Turn new track
85AF  AD 74 02   LDA $0274      Get length from command string,
85B2  C9 07      CMP #$07       and compare to 7 characters
85B4  90 32      BCC $85E8      Does cmd string have min. 7chars?
85B6  A5 06      LDA $06        YES—Get last track number and
85B8  A8         TAY            save it
85B9  E9 01      SBC #$01       Get current head position, above,
85BB  0A         ASL A          then calculate in half-steps and
85BC  85 64      STA $64        set it
```

```
85BE  C0 24     CPY #$24       Last track on side 2?
85C0  08        PHP            Save result of the test
85C1  AC 06 02  LDY $0206      Get 7th char from command string,
85C4  84 22     STY $22        and set as current track
85C6  88        DEY            From that, calculate and set
85C7  84 67     STY $67        target track -1
85C9  C0 23     CPY #$23       Is new track on side 2 of disk?
85CB  6A        ROR A          Move result in Bit 7
85CC  28        PLP            Get previous result again, & get
85CD  29 80     AND #$80       the last result ready again
85CF  90 0B     BCC $85DC      Last track on side 2 (Bit =1)?
85D1  30 12     BMI $85E5      YES—New track on side 2 (Bit =1)?
85D3  18        CLC            NO-- Compute and
85D4  A5 67     LDA $67        set new track
85D6  69 23     ADC #$23       on side
85D8  85 67     STA $67        2
85DA  30 09     BMI $85E5      Jump
85DC1 10 07     BPL $85E5      to $85E5
85DE  38        SEC            Compute new track
85DF  A5 67     LDA $67        number on side
85E1  E9 23     SBC #$23       1 and
85E3  85 67     STA $67        save it
85E53 4C BA 87  JMP $87BA      Turn track
85E81 60        RTS            Return from this subroutine
-----------------------------------------------------------------
[8381/83BE/8464/84EB/8509/85A7/8D64/8DB1/8EA3]
Perpare error byte for output
85E9  86 46     STX $46        Save error number
85EB  A5 5E     LDA $5E        Get command status byte and
85ED  29 F0     AND #$F0       isolate flag
85EF  05 46     ORA $46        Take up error # and set value
85F1  85 5E     STA $5E        as new status; also, character
85F3  85 46     STA $46        to be given
85F5  60        RTS            Return from this subroutine
-----------------------------------------------------------------
[8603]  Send byte over 1571 bus
85F6  20 59 EA  JSR $EA59      Test for ATN command mode
-----------------------------------------------------------------
[83C9/83D7/8467/84FD/853A/8548/854F/8556/855D/8563/8589/85FF/8609]
[8DBF/8DD4/8EA6]
Send byte over 1571 bus
85F9  AD 00 18  LDA $1800      Get bus control register;wait til
85FC  CD 00 18  CMP $1800      line status remains constant
85FF  D0 F8     BNE $85F9      Constant value applied?
8601  29 FF     AND #$FF       YES—Processor flag reset
8603  30 F1     BMI $85F6      Is ATN set?
8605  45 37     EOR $37        NO—Get bus status and test
```

```
8607   29 04      AND #$04       flag for Clock
8609   F0 EE      BEQ $85F9      Is Clock set?
860B   A5 46      LDA $46        YES—Get char. tobe sent & transfr
860D   8D 0C 40   STA $400C      to the serial output register
8610   A5 37      LDA $37        Flag for Clock; get and
8612   49 04      EOR #$04       invert
8614   85 37      STA $37        Store flag again
8616   A9 08      LDA #$08       Test bitflag for 'Register
8618¹  2C 0D 40   BIT $400D      output' and verify
861B   F0 FB      BEQ $8618      Is byte completely output?
861D   60         RTS            YES—Return from this subroutine
-------------------------------------------------------------------
[83E2/847E]
Calculate number of next IBM-34 sector
861E   AD 03 02   LDA $0203      Get track # from command string &
8621   C9 24      CMP #$24       compare with max. track +1
8623   90 02      BCC $8627      Is track on side 2?
8625   E9 23      SBC #$23       YES—Compute and set
8627¹  AA         TAX            track from side 1
8628   BD 2B 94   LDA $942B,X    Determine and save # of sectors
862B   AA         TAX            per track; from this,
862C   CA         DEX            get maximum sector number and
862D   86 46      STX $46        save it
862F   18         CLC            Set new sector number:
8630   AD 04 02   LDA $0204      Sector number from command string
8633   65 3C      ADC $3C        Compute sector format
8635   C5 46      CMP $46        Compare with maximum number
8637   90 0A      BCC $8643      Has legal range been exceeded?
8639   E5 46      SBC $46        YES—Set number of legal range
863B   F0 04      BEQ $8641      Last sector chosen?
863D   38         SEC            YES—Then calculate sector number
863E   E9 01      SBC #$01       (since sector 0 also exists)
8640   2C         .byte $2C      Jump to next 2 bytes(bit command)
8641¹  A5 46      LDA $46        Get first value computed and
8643¹  8D 04 02   STA $0204      save current sector number
8646   A9 88      LDA #$88       'Read sector from current track'
8648   85 5F      STA $5F        given as current jobcode
864A   60         RTS            Return from this subroutine
-------------------------------------------------------------------
[910E]
Execute job
864B   A6 F9      LDX $F9        Current buffer number
864D   08         PHP            Retain processor status
864E   58         CLI            Enable bus/controller interrupt
864F   20 B6 9F   JSR $9FB6      Start job loop and execute job
8652   C9 02      CMP #$02       Compare return message with 'Ok'
8654   90 05      BCC $865B      Job run error-free?
```

```
8656   20 83 86   JSR $8683       NO—Continue trying
8659   B5 00      LDA $00,X       Get and save return
865B¹  AA         TAX             message
865C   28         PLP             Re-establish processor status
865D   60         RTS             Return from this subroutine
-------------------------------------------------------------------
[83BA/845D/84AE]
Start job loop and execute job for buffer 0
865E   A2 00      LDX #$00        Determine buffer number
8660   08         PHP             Save processor status
8661   78         SEI             Disable bus/controller interrupt
8662   AD 00 1C   LDA $1C00       Get drive control register and
8665   09 08      ORA #$08        set bit for LED
8667   8D 00 1C   STA $1C00       LED on
866A   58         CLI             Enable bus/controller interrupt
866B   20 B6 9F   JSR $9FB6       Start job loop and execute job
866E   C9 02      CMP #$02        Test return message for 'OK'
8670   90 03      BCC $8675       Job run error-free?
8672   20 83 86   JSR $8683       NO-- Execute new attempt
8675¹  78         SEI             Disable bus/controller interrupt
8676   AD 00 1C   LDA $1C00       Get control register and
8679   29 F7      AND #$F7        clear LED bit
867B   8D 00 1C   STA $1C00       LED off
867E   B5 00      LDA $00,X       Get return message of last try,
8680   AA         TAX             and save it
8681   28         PLP             Re-establish processor status
8682   60         RTS             Return from this subroutine
-------------------------------------------------------------------
[8656/8672]
Continue attempt at job execution
8683   A9 FF      LDA #$FF        Set flag for
8685   8D 98 02   STA $0298       'Error in job execution'
8688   86 F9      STX $F9         Save buffer number of job
868A   AD 02 02   LDA $0202       Get jobcode and save
868D   85 5F      STA $5F         as current
868F   8D 4D 02   STA $024D       jobcode
8692   9D 5B 02   STA $025B,X     Arrange jobcode of buffer & give
8695   85 00      STA $00         to job loop as jobcode for
8697   20 B6 9F   JSR $9FB6       buffer 0
869A   4C 99 D5   JMP $D599       Control job execution
-------------------------------------------------------------------
869D1  20 59 EA   JSR $EA59       Test for ATN command mode
-------------------------------------------------------------------
```

```
[846A/86A6/86B0/8EA9]
Wait for jump of Clock input
86A0  AD 00 18   LDA $1800      Get bus control register and
86A3  CD 00 18   CMP $1800      compare w/value from 2nd reading
86A6  D0 F8      BNE $86A0      Is status of register constant?
86A8  29 FF      AND #$FF       Set processor flag
86AA  30 F1      BMI $869D      Is ATN set?
86AC  45 37      EOR $37        NO—Compare Clock value w/value of
86AE  29 04      AND #$04       the last call of this routine
86B0  F0 EE      BEQ $86A0      Has it been altered?
86B2  A5 37      LDA $37        YES—Then get flag and
86B4  49 04      EOR #$04       also re-save
86B6  85 37      STA $37        reversed flag
86B8  60         RTS            Return from this subroutine
-------------------------------------------------------------------
[Table will be used in 86E9]
Control bytes for functions before call of IBM-34 routines

Function of individual bits (Bit=1 called'Function activated'):
bit0  Error by job execution not given
bit1  Read next header and set head to last-read track
bit2  Wait until motor runs & head is in position (track and side)
bit3  Position head to new track ($67)
bit4  Drive motor on
bit5  Check write protect
bit6  Take sector number from command string
bit7  Set value for new track ($67)

86B9  00      %00000000  No status functions
86BA  15      %00010101  Motor on/wait/no error message
86BB  00      %00000000  No status functions
86BC  00      %00000000  No status functions
86BD  00      %00000000  No status functions
86BE  15      %00010101  Motor on/wait/no error message
86BF  00      %00000000  No status functions
86C0  BC      %10111100  Track/test WP/head/wait
86C1  34      %00110100  Test WP/motor on/wait
86C2  DE      %11011110  Track/sector/motor on/head/header
86C3  FE      %11111110  Track/sector/test WP/motor on/head/header
86C4  DC      %11011100  Track/sector/Motor on/head/wait
86C5  15      %00010101  Motor on/wait/no error message
86C6  15      %00010101  Motor on/wait/no error message
86C7  00      %00000000  No status functions
-------------------------------------------------------------------
```

```
[Table will be used in 873F]
Diskette routine addresses in IBM System-34 format
86C8  EC 89       0 / 00      $89EC  Execute reset ($EAA0)
86CA  EF 89       1 / 01      $89EF  Reset head to track
86CC  FD 89       2 / 02      $89FD  Test 'Write Protect'
86CE  03 8A       3 / 03      $8A03  Take up track parameters
86D0  08 8A       4 / 04      $8A08  No function (rts)
86D2  09 8A       5 / 05      $8A09  Read next IBM-34 header
86D4  BA 87       6 / 06      $87BA  No function (rts)
86D6  86 8A       7 / 07      $8A86  Format 'IBM Syst34' track
86D8  57 8C       8 / 08      $8C57  Format IBM-34 diskette
86D9  67 8D       9 / 09      $8D67  Read IBM-34 sector
86DC  F6 8D      10 / 0A      $8DF6  Write IBM-34 sector
86DE  C6 8E      11 / 0B      $8EC6  IBM-34 sector verify
86E0  18 8F      12 / 0C      $8F18  Test IBM34 sectr: empty byte
86E2  5F 8F      13 / 0D      $8F5F  Send IBM-34 sector sequence
86E4  B3 89      14 / 0E      $89B3  Initialize IBM-34 track
-----------------------------------------------------------------
[839D/8408/8499/84C5/8520/BF4E]
Routine to call IBM system 34 functions (number in accumulator)
86E6  78         SEI          Disable bus/controller interrupt
86E7  48         PHA          Save # of routine to be called
86E8  AA         TAX          and get corresponding
86E9  BD B9 86   LDA $86B9,X  control byte of routine
86EC  85 1B      STA $1B      and store it
86EE  A5 5E      LDA $5E      Set flag for IBM-34 format
86F0  09 80      ORA #$80     in command status
86F2  85 5E      STA $5E      byte
86F4  06 1B      ASL $1B      Test Bit7 of control byte
86F6  90 05      BCC $86FD    Set?
86F8  AD 03 02   LDA $0203    YES—Get # of track to be ctrolled
86FB  85 67      STA $67      and set pointers
86FD1 06 1B      ASL $1B      Test Bit6 of control byte
86FF  90 05      BCC $8706    Set?
8701  AD 04 02   LDA $0204    YES—Get number of desired sector
8704  85 43      STA $43      and save it
87061 06 1B      ASL $1B      Test Bit5 of control byte
8708  90 11      BCC $871B    Set?
870A  AD 00 1C   LDA $1C00    Get bus control register and test
870D  29 10      AND #$10     Bit for 'Write Protect'
870F  D0 0A      BNE $871B    Is there a write-protect tab?
8711  A5 3B      LDA $3B      YES—Set
8713  09 08      ORA #$08     'Write protect in place'
8715  85 3B      STA $3B      flag
8717  A2 08      LDX #$08     Save error #: 'Write Protect On'
8719  86 46      STX $46      as character to be given
871B2 06 1B      ASL $1B      Test Bit4 of control byte
```

```
871D   90 03      BCC $8722       Set?
871F   20 94 87   JSR $8794       YES—Drive motor on
8722¹  06 1B      ASL $1B         Test Bit 3 of control byte
8724   90 03      BCC $8729       Set?
8726   20 BA 87   JSR $87BA       YES—Turn target track from ($67)
8729¹  06 1B      ASL $1B         Test bit 2 of control byte
872B   90 03      BCC $8730       Set?
872D   20 B0 87   JSR $87B0       Wait until head & motor are ready
8730¹  20 54 89   JSR $8954       Activate head on current side
8733   06 1B      ASL $1B         Test Bit1 of control byte
8735   90 03      BCC $873A       Set?
8737   20 2A 89   JSR $892A       YES—Read IBM-34 header & set head
873A¹  A9 00      LDA #$00        Clear processor status register
873C   68         PLA             Get # of program to be called
873D   0A         ASL A           and double it (address table
873E   AA         TAX             takes 2-byte pointers)
873F   BD C8 86   LDA $86C8,X     Get program addr(lo-byte) from
8742   85 6F      STA $6F         table and set in pointer
8744   BD C9 86   LDA $86C9,X     Get high-byte and
8747   85 70      STA $70         save it
8749   20 61 87   JSR $8761       Execute program
874C   20 8F F9   JSR $F98F       Motor off (set flag)
874F   AE B0 01   LDX $01B0       Get return message of last job &
8752   E0 02      CPX #$02        compare with 'Ok'
8754   08         PHP             Save result
8755   06 1B      ASL $1B         Test bit0 of control byte
8757   B0 06      BCS $875F       Set?
8759   28         PLP             NO—Get result of eror check
875A   90 04      BCC $8760       Is job running error-free?
875C   4C 8C 83   JMP $838C       NO—Display error and end
875F¹  28         PLP             Prepare result of error check
8760¹  60         RTS             Return from this subroutine
8761¹  6C 6F 00   JMP ($006F)     Execute IBM 34 routine
---------------------------------------------------------------------
[87A3/99E7/A642/BF51]
Drive motor on
8764   08         PHP             Retain processor status
8765   78         SEI             Disable bus/controller interrupt
8766   AD 00 1C   LDA $1C00       Get bus control register and
8769   09 04      ORA #$04        set bit for 'Motor on'
876B   8D 00 1C   STA $1C00       Store register again
876E   28         PLP             Re-establish processor status
876F   60         RTS             Return from this subroutine
---------------------------------------------------------------------
```

```
[99FB/9A39/A654/BF54]
Drive motor off
8770  08         PHP          Retain processor status
8771  78         SEI          Disable bus/controller interrupt
8772  AD 00 1C   LDA $1C00    Get bus control register and
8775  29 FB      AND #$FB     clear bit for 'Motor on'
8777  8D 00 1C   STA $1C00    Reset control register
877A  28         PLP          Re-establish processor status
877B  60         RTS          Return from this subroutine
-------------------------------------------------------------
[884F]  cf. C100/C118
Drive LED on
877C  08         PHP          Retain processor status
877D  78         SEI          Disable bus/controller interrupt
877E  AD 00 1C   LDA $1C00    Get bus control register
8781  09 08      ORA #$08     and set bit for 'LED on'
8783  8D 00 1C   STA $1C00    Store register again
8786  28         PLP          Re-establish processor status
8787  60         RTS          Return from this subroutine
-------------------------------------------------------------
[8861]
Drive LED off
8788  08         PHP          Retain processor status
8789  78         SEI          Disable bus/controller interrupt
878A  AD 00 1C   LDA $1C00    Get bus control register
878D  29 F7      AND #$F7     and clear bit for 'LED on'
878F  8D 00 1C   STA $1C00    Set control register again
8792  28         PLP          Re-establish processor status
8793  60         RTS          Return from this subroutine
-------------------------------------------------------------
[871F]
Motor on and initialize flag
8794  08         PHP          Retain processor status
8795  78         SEI          Disable bus/controller interrupt
8796  A5 20      LDA $20      Get drive status and
8798  C9 20      CMP #$20     test for 'Motor running'
879A  F0 0E      BEQ $87AA    Is the motor already active?
879C  AD 02 02   LDA $0202    NO-Get jobcode of routine and
879F  29 01      AND #$01     determine drive desired
87A1  85 3E      STA $3E      Set number as current drive
87A3  20 64 87   JSR $8764    Motor on
87A6  A9 A0      LDA #$A0     Drive status at
87A8  85 20      STA $20      'Motor at/ not at turn number'
87AA1 A9 32      LDA #$32     Set counter for motor
87AC  85 48      STA $48      runtime
87AE  28         PLP          Get status register again
87AF  60         RTS          Return from this subroutine
```

```
[872D]
Wait until motor is set to turn number, and head is set in position
87B0   08          PHP             Retain status
87B1   58          CLI             Enable bus/controller interrupt
87B2¹  A5 20       LDA $20         Get drive status and test for
87B4   C9 20       CMP #$20        'no stepper / drive ready'
87B6   D0 FA       BNE $87B2       Drive ready?
87B8   28          PLP             YES—Re-establish status
87B9   60          RTS             Return from this subroutine
-------------------------------------------------------------------
[85E5/8726/8927/894C/89FA/8D0A/8D3A/8D51/8F6D]
Turn to new track
87BA   08          PHP             Retain status
87BB   58          CLI             Enable bus/controller interrupt
87BC   A5 67       LDA $67         Get # of new track & compute the
87BE   0A          ASL A           number of absolute half-steps
87BF   C5 64       CMP $64         Compare w/current head position
87C1   F0 1A       BEQ $87DD       Identical?
87C3²  A5 67       LDA $67         NO—Get number of new track and
87C5   0A          ASL A           compute half-steps
87C6   C5 64       CMP $64         Compare w/current counter status
87C8   F0 0E       BEQ $87D8       Identical?
87CA   B0 06       BCS $87D2       NO—currenut cntr>target position?
87CC   20 E7 87    JSR $87E7       YES—Move a half-step out until
87CF   4C C3 87    JMP $87C3       track is reached
87D2¹  20 DF 87    JSR $87DF       Move one half-step out until
87D5   4C C3 87    JMP $87C3       track is reached
87D8¹  A0 10       LDY #$10        Initialize counter
87DA   20 29 88    JSR $8829       40/20 ms delay (1/2 mHz)
87DD¹  28          PLP             Re-establish status
87DE   60          RTS             Return from this subroutine
-------------------------------------------------------------------
[87D2]
One half-step in
87DF   A5 64       LDA $64         Determine current position
87E1   18          CLC             Prepare addition
87E2   69 01       ADC #$01        Add a half-step
87E4   4C 14 88    JMP $8814       Control stepper motor
-------------------------------------------------------------------
[87E7]  cf. 9A66/FF45
One half-step out
87E7   A0 63       LDY #$63        # of scan attempts/track 0 (99)
87E9¹  AD 0F 18    LDA $180F       Get control register A
87EC   6A          ROR A           Track 0 identifier(bit 0)in carry
87ED   08          PHP             and save carry
87EE   AD 0F 18    LDA $180F       Read control register again
87F1   6A          ROR A           Shift track 0 identifier (bit 0)
```

```
87F2   6A          ROR A          to bit 7
87F3   28          PLP            Get previous scan result
87F4   29 80       AND #$80       Isolate last scan result
87F6   90 04       BCC $87FC      Track0 active in 1st test(bit=0)?
87F8   10 15       BPL $880F      NO—Is track 0 now reached?
87FA   30 02       BMI $87FE      YES—Jump to $877C
87FC1  30 11       BMI $880F      Track 0 still active?
------------------------------------------------------------------
Status of track 0 write-protection has not changed
87FE1  88          DEY            YES—Take another look
87FF   D0 E8       BNE $87E9      All attempts already performed?
8801   B0 0C       BCS $880F      YES—Is head at track 0 position?
8803   AD 00 1C    LDA $1C00      YES—Control register f/step-motor
8806   29 03       AND #$03       Isolate step bits
8808   D0 05       BNE $880F      Is stepper reel being controlled?
880A   A9 00       LDA #$00       Clear current
880C   85 64       STA $64        head position
880E   60          RTS            Return from this subroutine
------------------------------------------------------------------
Track 0 write-protection status HAS changed
880F3  A5 64       LDA $64        Get current head position and
8811   38          SEC            limit to one
8812   E9 01       SBC #$01       half-step
88141  85 64       STA $64        Save new position
8816   29 03       AND #$03       Set up and save control bits for
8818   85 6F       STA $6F        stepper reel
881A   08          PHP            Retain processor status
881B   78          SEI            Disable bus/controller interrupt
881C   AD 00 1C    LDA $1C00      Get control register
881F   29 FC       AND #$FC       Mask out stepper control and set
8821   05 6F       ORA $6F        new bit values
8823   8D 00 1C    STA $1C00      New value in control register
8826   28          PLP            Re-establish processor status
8827   A0 06       LDY #$06       Set counter for 13/7.5 ms delay
88292  20 30 88    JSR $8830      Approx. 2.6/1.3 ms
882C   88          DEY            Adjust counter
882D   D0 FA       BNE $8829      Delay already run up?
882F   60          RTS            YES—Return from this subroutine
------------------------------------------------------------------
```

```
[8829]
Approx. 2.6/1.3 ms delay (2583 cycles until resumption point)
8830   A2 02       LDX #$02          Number of counter loops
8832   A9 00       LDA #$00          Initialize pointer
8834²  69 01       ADC #$01          and increase by one
8836   D0 FC       BNE $8834         Already counted to 256?
8838   CA          DEX               YES—Next count loop
8839   D0 F9       BNE $8834         All loops done?
883B   60          RTS               YES—Return from this subroutine
-------------------------------------------------------------------
[8A4D/8DAE/8E9D]
Get error from CP/M controller
883C   EA          NOP               Delay until controller is ready
883D   AD 00 20    LDA $2000         Read status register
8840   4A          LSR A             Shft error bits:'Record not found
8841   4A          LSR A             and 'CRC Error' (Bit 3 and 4) in
8842   4A          LSR A             positions 0 and 1
8843   29 03       AND #$03          Isolate error bits and
8845   AA          TAX               set up error pointer
8846   BD 82 8A    LDA $8A82,X       Determine and set number of error
8849   8D B0 01    STA $01B0         message
884C   AA          TAX               Save error number
884D   60          RTS               Return from this subroutine
-------------------------------------------------------------------
[89C0/8A30/8A88/8D85/8E63/8EDE/8F27]
Send command over CP/M controller (WD 1770)
884E   48          PHA               Save command
884F   20 7C 87    JSR $877C         Switch on LED on drive
8852   68          PLA               Repeat command
8853   8D 00 20    STA $2000         and send on CP/M controller
8856   A9 01       LDA #$01          Bit for 'Busy - Flag'
8858   EA          NOP               Delay until controller is ready
8859¹  2C 00 20    BIT $2000         Test controller status register
885C   F0 FB       BEQ $8859         Is command taken (busy set)?
885E   4C 7E A4    JMP $A47E         YES—Wait 45 cycles
-------------------------------------------------------------------
[89C3/8A4A/8C48/8DAB/8F15/8F5C]
Wait until current command of CP/M controller is done
8861   20 88 87    JSR $8788         LED off
8864   A9 01       LDA #$01          Test bit for 'Busy - Flag'
8866¹  2C 00 20    BIT $2000         in status register
8869   D0 FB       BNE $8866         Is command still active?
886B   60          RTS               NO—Return from this subroutine
-------------------------------------------------------------------
```

```
[8DED/886C]
Compute number of next sector
886C  A5 60     LDA $60       Get smallest sector number and
886E  38        SEC           format until
886F  E9 01     SBC #$01      sector number reaches zero;
8871  85 46     STA $46       save it
8873  AD 04 02  LDA $0204     Get # of current sector; from
8876  18        CLC           that, add
8877  65 3C     ADC $3C       sector format
8879  C5 61     CMP $61       Compare with maximum sector #
887B  F0 07     BEQ $8884     Is new number identical?
887D  90 05     BCC $8884     NO-Is new number smaller?
887F  E5 61     SBC $61       NO-Calculate sector number from
8881  18        CLC           allowable range; note
8882  65 46     ADC $46       common sector shifts
8884² 8D 04 02  STA $0204     Set new sector number
8887  60        RTS           Return from this subroutine
-----------------------------------------------------------
[8CF9]
Make table of sector numbers available for formatting
8888  A0 00     LDY #$00      Clear pointer to current sector #
888A  A2 00     LDX #$00      Clear sector counter
888C  AD 03 02  LDA $0203     Limit number of first
888F  29 3F     AND #$3F      sector to range of
8891  8D 03 02  STA $0203     0-63 and save
8894  85 60     STA $60       as smallest sector
8896  48        PHA           Save sector number
8897  AD 07 02  LDA $0207     Retain number of last
889A  48        PHA           sector
889B  EE 04 02  INC $0204     Set up sector format
889E¹ AD 03 02  LDA $0203     Get number of current sector and
88A1  99 0B 02  STA $020B,Y   insert in table
88A4  EE 03 02  INC $0203     Set number of next sector
88A7  E8        INX           Number of sectors set up
88A8  98        TYA           Get pointer to sector position
88A9  18        CLC           and compute sector
88AA  6D 04 02  ADC $0204     format
88AD  A8        TAY           Save new pointer and compare with
88AE  C0 20     CPY #$20      max. sector number
88B0  B0 0C     BCS $88BE     Gone over 32?
88B2  CC 07 02  CPY $0207     NO-Test number of last sector
88B5  90 1A     BCC $88D1     Reached this number?
88B7  D0 12     BNE $88CB     NO-Last number reached?
88B9  EC 07 02  CPX $0207     YES-Test # of sectors set out
88BC  F0 0D     BEQ $88CB     All sectors made available?
88BE² CE 04 02  DEC $0204     NO-Adjust sector format
88C1  68        PLA           Re-establish maximum
```

```
88C2   8D 07 02   STA $0207       sector number
88C5   68         PLA             Get number of first sector and
88C6   8D 03 02   STA $0203       set it
88C9   38         SEC             Flag for 'error  encountered'
88CA   60         RTS             Return from this subroutine
88CB2  98         TYA             Compute pointer/current sector
88CC   38         SEC             position in allowable
88CD   ED 07 02   SBC $0207       sector range
88D0   A8         TAY             and save it
88D11  EC 07 02   CPX $0207       Test number of sectors set out
88D4   D0 C8      BNE $889E       All sector #'s already in table?
88D6   86 97      STX $97         YES—save number of sectors
88D8   CA         DEX             Set up number of
88D9   8A         TXA             last sector
88DA   18         CLC             Compute # of smallest sector from
88DB   65 60      ADC $60         that, and save as number of
88DD   85 61      STA $61         largest sector
88DF   C5 60      CMP $60         Compare with smaller number
88E1   90 DB      BCC $88BE       Has sector been set out?
88E3   68         PLA             NO—Re-establish
88E4   8D 07 02   STA $0207       maximum sector number
88E7   68         PLA             Get number of first sector
88E8   8D 03 02   STA $0203       and set it
88EB   CE 04 02   DEC $0204       Adjust sector format
88EE   18         CLC             Flag for 'no error found'
88EF   60         RTS             Return from this subroutine
--------------------------------------------------------------------
[8D1F]
Test CP/M-sectors after formatting for empty bytes
88F0   AD B0 01   LDA $01B0       Save current track for
88F3   48         PHA             formatting
88F4   A0 00      LDY #$00        Clear counter for current sector
88F6   84 24      STY $24         number
88F81  A4 24      LDY $24         Get number of current sector and
88FA   B9 0B 02   LDA $020B,Y     determine sector number of header
88FD   8D 02 20   STA $2002       Send sector over CP/M controller
8900   20 18 8F   JSR $8F18       Test sector
8903   AE B0 01   LDX $01B0       Get return message
8906   E0 02      CPX #$02        and compare w/ value for 'Ok'
8908   B0 0B      BCS $8915       Is there an error here?
890A   E6 24      INC $24         NO—Choose next sector
890C   A4 24      LDY $24         Get current sector # and compare
890E   CC 07 02   CPY $0207       with maximum amount
8911   D0 E5      BNE $88F8       All sectors already checked?
8913   18         CLC             YES—Set flag for 'Ok'
8914   24         .byte $24       Jump to next byte (bit-command)
89151  38         SEC             Set flag for error
```

```
8916  68          PLA           Reset current track number of
8917  8D B0 01    STA $01B0     format procedure
891A¹ 60          RTS           Return from this subroutine
-------------------------------------------------------------------
[8DF3/8EC2]
Set next track
891B  AD 74 02    LDA $0274     Test length/command string in
891E  C9 07       CMP #$07      input buffer against 7
8920  90 F8       BCC $891A     Is command less than 7 chars.?
8922  AD 06 02    LDA $0206     NO—Get 7th character and take up
8925  85 67       STA $67       as current target track
8927  4C BA 87    JMP $87BA     Control track
-------------------------------------------------------------------
[8737]
Read next IBM system 34 sector and set head accordingly
892A  AD B0 01    LDA $01B0     Keep current error return
892D  48          PHA           message
892E  20 27 8A    JSR $8A27     Read next IBM-34 header
8931  AE B0 01    LDX $01B0     Get return message and check for
8934  E0 02       CPX #$02      error message
8936  90 0D       BCC $8945     Header been read error-free?
8938  20 EF 89    JSR $89EF     YES—Set head to track 0
893B  20 27 8A    JSR $8A27     Read next header
893E  AE B0 01    LDX $01B0     Get return message
8941  E0 02       CPX #$02      and test for error message
8943  B0 0A       BCS $894F     Header been read error-free?
8945¹ A5 67       LDA $67       YES—Get # of current target track
8947  0A          ASL A         and determine number of steps
8948  C5 64       CMP $64       Compare with current position
894A  F0 03       BEQ $894F     Is track already reached?
894C  20 BA 87    JSR $87BA     NO—Set head to target track
894F² 68          PLA           Repeat previous error number
8950  8D B0 01    STA $01B0     and set
8953  60          RTS           Return from this subroutine
-------------------------------------------------------------------
[8730/8CD5]
Activate head at current diskette side
8954  08          PHP           Retain processor status
8955  78          SEI           Disable bus/controller interrupt
8956  A5 3B       LDA $3B       Get flag from
8958  29 10       AND #$10      command number
895A  C9 10       CMP #$10      Take flag (bit 4) in carry
895C  20 F3 93    JSR $93F3     Set head to chosen side
895F  28          PLP           Re-establish processor status
8960  60          RTS           Return from this subroutine
-------------------------------------------------------------------
```

```
[852A]
Determine smallest and greatest sector numbers
8961   A4 97      LDY $97       Number of sectors laid out
8963   88         DEY           Counter to last sector position
8964   A9 FF      LDA #$FF      Maximum possible number;
8966¹  D9 0B 02   CMP $020B,Y   Compare with sector number
8969   90 03      BCC $896E     Is sector number less?
896B   B9 0B 02   LDA $020B,Y   YES—Get new sector number and
896E¹  88         DEY           set pointer to next sector naming
896F   10 F5      BPL $8966     All sectors already checked out?
8971   85 60      STA $60       YES—Set smallest sector number
8973   A4 97      LDY $97       Number of sectors laid out
8975   88         DEY           Counter to last sector position
8976   A9 00      LDA #$00      Smallest value
8978¹  D9 0B 02   CMP $020B,Y   Compare with sector number?
897B   B0 03      BCS $8980     Is number greater?
897D   B9 0B 02   LDA $020B,Y   YES—Take new sector number
8980¹  88         DEY           Pointer to next sector naming
8981   10 F5      BPL $8978     All sectors already checked?
8983   85 61      STA $61       YES—Save greatest sector number
8985   60         RTS           Return from this subroutine
-----------------------------------------------------------------
[852D]
Compute sector format from sector sequence
8986   A6 97      LDX $97       Number of sectors in table
8988   A0 00      LDY #$00      Reset position pointer
898A¹  B9 0B 02   LDA $020B,Y   Get sector # from table & compare
898D   C5 60      CMP $60       with smallest number
898F   F0 05      BEQ $8996     Identical?
8991   C8         INY           NO—Pointer to next sector
8992   C4 97      CPY $97       Compare with # of sector numbers
8994   D0 F4      BNE $898A     Already tested?
8996¹  84 5F      STY $5F       YES—Save place of smallest sector
8998   A5 60      LDA $60       Get smallest sector number
899A   18         CLC           and draw up number of next
899B   69 01      ADC #$01      sector
899D   85 46      STA $46       Save number
899F   A2 FF      LDX #$FF      Initialize cntr for sector format
89A1²  B9 0B 02   LDA $020B,Y   Get sector # from table and
89A4   C5 46      CMP $46       compare with second sector
89A6   F0 0A      BEQ $89B2     Identical?
89A8   E8         INX           NO—Increment sector format
89A9   C8         INY           Pointer to next sector number
```

```
89AA  C4 97      CPY $97       Test against number of sectors
89AC  D0 F3      BNE $89A1     All sectors handled?
89AE  A0 00      LDY #$00      YES-Reset pointers
89B0  F0 EF      BEQ $89A1     Jump to $89A1
89B2¹ 60         RTS           Return from this subroutine
-----------------------------------------------------------------
[8A0C/8CDE]
Initialize CP/M controller to track
89B3  A5 6F      LDA $6F       Hold zeropage area to be used
89B5  48         PHA           for temporary storage
89B6  08         PHP           Save processor status
89B7  78         SEI           Disable bus/controller interrupt
89B8  AD 01 20   LDA $2001     Set current track # as track
89BB  8D 03 20   STA $2003     to be newly initialized
89BE  A9 18      LDA #$18      %00011000 'Seek' (set track)
89C0  20 4E 88   JSR $884E     command on CP/M controller
89C3  20 61 88   JSR $8861     Wait until command is executed
89C6  A2 00      LDX #$00      Clear counter for number of
89C8  A0 80      LDY #$80      tries
89CA  AD 00 20   LDA $2000     Read CP/M status register
89CD  29 02      AND #$02      Get flag for status of index hole
89CF  85 6F      STA $6F       and save it
89D1² AD 00 20   LDA $2000     Re-read status reg. & status of
89D4  29 02      AND #$02      index hole light box
89D6  C5 6F      CMP $6F       Compare with the former
89D8  F0 04      BEQ $89DE     Index hole found?
89DA  28         PLP           YES-Re-establish processor status
89DB  4C E7 89   JMP $89E7     Set index flag and end
89DE¹ CA         DEX           Counter for tries (low-byte)
89DF  D0 F0      BNE $89D1     Is counter finished?
89E1  88         DEY           YES-Decrement high-byte
89E2  D0 ED      BNE $89D1     Is counter finished?
89E4  28         PLP           YES-Re-establish processor status
89E5  38         SEC           Flag for 'Index hole not found'
89E6  24 18      .byte $24     Jump to next byte (bit command)
89E7¹ 18         CLC           Flag for 'Index hole found'
89E8  68         PLA           Re-arrange
89E9  85 6F      STA $6F       zero-page area
89EB  60         RTS           Return from this subroutine
-----------------------------------------------------------------
[Vector: 86C8]
89EC  4C A0 EA   JMP $EAA0     Execute 1571 reset
-----------------------------------------------------------------
```

```
[8938/8A09/8CDB/8CE8/8F61/Vector: 86CA]
Replace head at track 0
89EF  A9 B4      LDA #$B4       Set # of current halftrack
89F1  85 64      STA $64        steps for track 37
89F3  A9 00      LDA #$00       Place CP/M controller at
89F5  8D 01 20   STA $2001      track 0
89F8  85 67      STA $67        Set new target track
89FA  4C BA 87   JMP $87BA      Position head
-----------------------------------------------------------------
[Vector: 86CC]
Test status of write-protect  notch
89FD  AD 00 1C   LDA $1C00      Get drive control reg hole, get
8A00  29 10      AND #$10       bit f/'Write Protect'(low active)
8A02  60         RTS            Return from this subroutine
-----------------------------------------------------------------
[Vector: 86CF]
Set track parameters
8A03  84 67      STY $67        Set track to be controlled
8A05  86 64      STX $64        Curr.position in half-track steps
8A07  60         RTS            Return from this subroutine
-----------------------------------------------------------------
[Vector: 864E]
8A08  60         RTS            Return from this subroutine
-----------------------------------------------------------------
[Vector: 86D2]
Read header of next CP/M sector and in buffer $0024
8A09  20 EF 89   JSR $89EF      Set head to track 0
8A0C  20 B3 89   JSR $89B3      Initialize controller
8A0F  B0 0F      BCS $8A20      Index hole on hand?
8A11  20 27 8A   JSR $8A27      YES—Read header and set pointer
8A14  BD 7E 8A   LDA $8A7E,X    Get # of sectors to a track and
8A17  85 97      STA $97        save them
8A19  85 61      STA $61        Set as largest sector number
8A1B  A9 01      LDA #$01       Determine smallest
8A1D  85 60      STA $60        sector number
8A1F  60         RTS            Return from this subroutine
8A20[1] A9 0D    LDA #$0D       Set error message --
8A22  8D B0 01   STA $01B0      'Index not found'
8A25  D0 3E      BNE $8A65      Jump to $8A65
-----------------------------------------------------------------
[892E/893B/8A11/8F74/8F82]
Read next IBM System 34 header and set sector pointer
8A27  A9 00      LDA #$00       Clear pointer for
8A29  8D 71 02   STA $0271      # bytes per sector portion and
8A2C  85 44      STA $44        number of portions
8A2E  A9 C8      LDA #$C8       %11001000'Read address'(Readhead)
8A30  20 4E 88   JSR $884E      Command on CP/M controller
```

```
8A33   A2 00      LDX #$00      Clear buffer pointer
8A35   A0 06      LDY #$06      Number of header bytes
8A37²  AD 00 20   LDA $2000     Read status register and
8A3A   29 03      AND #$03      isolate flag
8A3C   4A         LSR A         Flag: 'Command in process' (Busy)
8A3D   90 0B      BCC $8A4A     Is command still active?
8A3F   F0 F6      BEQ $8A37     YES—Any more header data?
8A41   AD 03 20   LDA $2003     YES—Get data byte and write
8A44   95 24      STA $24,X     in header buffer
8A46   E8         INX           Set buffer pointer to next byte
8A47   88         DEY           Decrement number of header bytes
8A48   D0 ED      BNE $8A37     All bytes read?
8A4A¹  20 61 88   JSR $8861     YES—Wait until command is ended
8A4D   20 3C 88   JSR $883C     Get return message frm controller
8A50   A5 24      LDA $24       Get track # from header read and
8A52   0A         ASL A         compute number of half-steps
8A53   85 64      STA $64       Save as current head position
8A55   A5 27      LDA $27       Get identifier for sector size
--------------------------------------------------------------
[8C7B/8C9F]
Set pointer for sector type
8A57   29 03      AND #$03      Isolate significant bits
8A59   AA         TAX           and save value
8A5A   BD 72 8A   LDA $8A72,X   Get # of bytes per sector portion
8A5D   8D 71 02   STA $0271     and save it
8A60   BD 76 8A   LDA $8A76,X   Determine # of portions /sector
8A63   85 44      STA $44       and take up
8A65¹  A5 5E      LDA $5E       Get command status byte & isolate
8A67   29 80      AND #$80      flag for IBM-34 diskette
8A69   0D B0 01   ORA $01B0     Combine current track # and set
8A6C   1D 7A 8A   ORA $8A7A,X   identifier for sector length
8A6F   85 5E      STA $5E       Re-set command status byte
8A71   60         RTS           Return from this subroutine
--------------------------------------------------------------
[8A5A]  Number of bytes per sector portion
8A72   7F                       Value for  128 bytes / sector
8A73   FF                       Value for  256 bytes / sector
8A74   FF                       Value for  512 bytes / sector
8A75   FF                       Value for 1024 bytes / sector
--------------------------------------------------------------
```

```
[8A60]
Number of portions per CP/M sector
8A76  01                             Value for  128 bytes / sector
8A77  01                             Value for  256 bytes / sector
8A78  02                             Value for  512 bytes / sector
8A79  04                             Value for 1024 bytes / sector
-------------------------------------------------------------------
[8A6C]
Identifier for sector length (in most significant byte-half)
8A7A  00                             Value for  128 bytes / sector
8A7B  10                             Value for  256 bytes / sector
8A7C  20                             Value for  512 bytes / sector
8A7D  30                             Value for 1024 bytes / sector
-------------------------------------------------------------------
[8A14]
Number of sectors per track; number of highest sector
8A7E  1A                             Value for  128 bytes / sector
8A7F  10                             Value for  256 bytes / sector
8A80  09                             Value for  512 bytes / sector
8A81  05                             Value for 1024 bytes / sector
-------------------------------------------------------------------
[8846]
CP/M error messages
8A82  01                             Number for 'OK'
8A83  09                             Number for 'False checksum'
8A84  02                             # for 'Sector header not found'
8A85  03                             Number for 'Sync not found'
-------------------------------------------------------------------
[8D14]
Format CP/M track in 'IBM System 34 format'
8A86  A9 F8     LDA #$F8             %111110000Write'Write track'track
8A88  20 D0 87  JSR $87D0            Give command over CP/M controller
8A8B  24 3B     BIT $3B              Test flag in command number
8A8D  50 62     BVC $8AF1            Should track index be written?
-------------------------------------------------------------------
Write track-Index save (after index hole)
8A8F  A2 50     LDX #$50             YES-# of bytes f/index Pulse(80)
8A91² AD 00 20  LDA $2000            Get status register &
8A94  29 03     AND #$03             isolate command bits
8A96  4A        LSR A                Test bit for 'Busy'
8A97  90 60     BCC $8AF9            Should command be executed?
8A99  F0 F6     BEQ $8A91            YES-Data controller ready?
8A9B  A9 4E     LDA #$4E             Write byte value for Pre-Index 1
8A9D  8D 03 20  STA $2003            on diskette
8AA0  CA        DEX                  Write next byte
8AA1  D0 EE     BNE $8A91            All bytes already?
8AA3  A2 0C     LDX #$0C             YES-Set counter for spaces(12)
```

```
8AA5²  AD 00 20   LDA $2000      Get status register
8AA8   29 03      AND #$03       and isolate command bits
8AAA   4A         LSR A          Test bit for 'Busy'
8AAB   90 4C      BCC $8AF9      Should command still be executed?
8AAD   F0 F6      BEQ $8AA5      YES—Data controller ready?
8AAF   A9 00      LDA #$00       Write byte value for Pre-Index 2
8AB1   8D 03 20   STA $2003      to diskette
8AB4   CA         DEX            Write next byte
8AB5   D0 EE      BNE $8AA5      All bytes ready?
8AB7   A2 03      LDX #$03       YES—Set counter
8AB9²  AD 00 20   LDA $2000      Get status register &
8ABC   29 03      AND #$03       isolate command bits
8ABE   4A         LSR A          Test bit for 'Busy'
8ABF   90 38      BCC $8AF9      Will command still be executed?
8AC1   F0 F6      BEQ $8AB9      YES—Data controller ready?
8AC3   A9 F6      LDA #$F6       Write value for time byte $C2
8AC5   8D 03 20   STA $2003      to diskette
8AC8   CA         DEX            Write next byte
8AC9   D0 EE      BNE $8AB9      All bytes?
8ACB¹  AD 00 20   LDA $2000      Get status register &
8ACE   29 03      AND #$03       isolate command bits
8AD0   4A         LSR A          Test bit for 'Busy'
8AD1   90 26      BCC $8AF9      Will command still be executed?
8AD3   F0 F6      BEQ $8ACB      YES—Data controller ready?
8AD5   A9 FC      LDA #$FC       Wrte byte val:"Addres Index Save'
8AD7   8D 03 20   STA $2003      to diskette
8ADA   A2 32      LDX #$32       Set counter (50)
8ADC   EA         NOP            Two cycles delay
8ADD²  AD 00 20   LDA $2000      Get status register & isolate
8AE0   29 03      AND #$03       command bits
8AE2   4A         LSR A          Test bit for 'Busy'
8AE3   90 14      BCC $8AF9      Will command  still be executed?
8AE5   F0 F6      BEQ $8ADD      YES—Data controller ready?
8AE7   A9 4E      LDA #$4E       Write byte value for Post-Index
8AE9   8D 03 20   STA $2003      to diskette
8AEC   CA         DEX            Write next byte
8AED   D0 EE      BNE $8ADD      All bytes ready?
8AEF   F0 14      BEQ $8B05      YES—Jump to $8B05
-----------------------------------------------------------
[8A8D]  Format sectors
8AF1   A2 3C      LDX #$3C       Set counter (60)
8AF3²  AD 00 20   LDA $2000      Get status register AND isolate
8AF6   29 03      AND #$03       command bits
8AF8   4A         LSR A          Test bit for 'Busy'
8AF9⁵  90 28      BCC $8B23      Will command  still be executed?
8AFB   F0 F6      BEQ $8AF3      YES—Data controller ready?
8AFD   A9 4E      LDA #$4E       Write byte value for space 1
```

```
8AFF   8D 03 20   STA $2003     to diskette
8B02   CA         DEX           Write next byte
8B03   D0 EE      BNE $8AF3     All bytes ready?
8B05¹  A0 01      LDY #$01      YES—Sector counter
8B07¹  A2 0C      LDX #$0C      Set counter
8B09²  AD 00 20   LDA $2000     Get status register & isolate
8B0C   29 03      AND #$03      command bits
8B0E   4A         LSR A         Test bit for 'Busy'
8B0F   90 12      BCC $8B23     Will command still be executed?
8B11   F0 F6      BEQ $8B09     YES—Data controller ready?
8B13   A9 00      LDA #$00      Write byte value for 2nd part of
8B15   8D 03 20   STA $2003     space 1 to diskette
8B18   CA         DEX           Write next byte
8B19   D0 EE      BNE $8B09     All bytes done?
8B1B   A2 03      LDX #$03      Set counter
8B1D²  AD 00 20   LDA $2000     Get status register & isolate
8B20   29 03      AND #$03      command bits
8B22   4A         LSR A         Test bit for 'Busy'
8B23²  90 57      BCC $8B7C     Will command still be executed?
8B25   F0 F6      BEQ $8B1D     YES—Data controller ready?
8B27   A9 F5      LDA #$F5      Write value time byte $A1
8B29   8D 03 20   STA $2003     to diskette
8B2C   CA         DEX           Write next byte
8B2D   D0 EE      BNE $8B1D     All bytes done up?
8B2F²  AD 00 20   LDA $2000     Get status register & isolate
8B32   29 03      AND #$03      command bits
8B34   4A         LSR A         Test bit for 'Busy'
8B35   90 45      BCC $8B7C     Will command still be executed?
8B37   F0 F6      BEQ $8B2F     YES—Data controller ready?
8B39   A9 FE      LDA #$FE      Write byte value:'ID Adress Save'
8B3B   8D 03 20   STA $2003     to diskette
8B3E¹  AD 00 20   LDA $2000     Get status register & isolate
8B41   29 03      AND #$03      command bits
8B43   4A         LSR A         Test bit for 'Busy'
8B44   90 36      BCC $8B7C     Will command still be executed?
8B46   F0 F6      BEQ $8B3E     YES—Data controller ready?
8B48   AD B0 01   LDA $01B0     Write current track number
8B4B   8D 03 20   STA $2003     to diskette
8B4E¹  AD 00 20   LDA $2000     Get status register & isolate
8B51   29 03      AND #$03      command bits
8B53   4A         LSR A         Test bit for 'Busy'
8B54   90 26      BCC $8B7C     Will command still be executed?
8B56   F0 F6      BEQ $8B4E     YES—Data controller ready?
8B58   A5 3B      LDA $3B       Get flag for current disk side &
8B5A   29 10      AND #$10      test it
8B5C   D0 03      BNE $8B61     Is side 1 active?
8B5E   A9 00      LDA #$00      YES—Then set side identifier
```

```
8B60   2C          .byte $2C       Jump to next 2 bytes(bit command)
8B61¹  A9 01       LDA #$01        Write side 2 identifier
8B63   8D 03 20    STA $2003       to diskette
8B66¹  AD 00 20    LDA $2000       Get status register & isolate
8B69   29 03       AND #$03        command bits
8B6B   4A          LSR A           Test bit for 'Busy'
8B6C   90 0E       BCC $8B7C       Will command still be executed?
8B6E   F0 F6       BEQ $8B66       YES-Data controller ready?
8B70   B9 0A 02    LDA $020A,Y     Write sector number
8B73   8D 03 20    STA $2003       to diskette
8B76¹  AD 00 20    LDA $2000       Get status register & isolate
8B79   29 03       AND #$03        command bits
8B7B   4A          LSR A           Test bit for 'Busy'
8B7C⁵  90 33       BCC $8BB1       Will command still be executed?
8B7E   F0 F6       BEQ $8B76       YES-Data controller ready?
8B80   AD 05 02    LDA $0205       Write identifier for
8B83   8D 03 20    STA $2003       sector length to diskette
8B86¹  AD 00 20    LDA $2000       Get status register & isolate
8B89   29 03       AND #$03        command bits
8B8B   4A          LSR A           Test bit for 'Busy'
8B8C   90 23       BCC $8BB1       Will command still be executed?
8B8E   F0 F6       BEQ $8B86       YES-Data controller ready?
8B90   A9 F7       LDA #$F7        Write byte value for 2 CRC bytes
8B92   8D 03 20    STA $2003       to diskette
8B95   A2 16       LDX #$16        Set counter (22)
8B97²  AD 00 20    LDA $2000       Get status register & isolate
8B9A   29 03       AND #$03        command bits
8B9C   4A          LSR A           Test bit for 'Busy'
8B9D   90 12       BCC $8BB1       Will command still be executed?
8B9F   F0 F6       BEQ $8B97       YES-Data controller ready?
8BA1   A9 4E       LDA #$4E        Write byte value for space 2
8BA3   8D 03 20    STA $2003       to diskette
8BA6   CA          DEX             Write next byte
8BA7   D0 EE       BNE $8B97       All bytes already?
8BA9   A2 0C       LDX #$0C        Set counter (12)
8BAB²  AD 00 20    LDA $2000       Get status register & isolate
8BAE   29 03       AND #$03        command bits
8BB0   4A          LSR A           Test bit for 'Busy'
8BB1³  90 38       BCC $8BEB       Will command still be executed?
8BB3   F0 F6       BEQ $8BAB       YES-Data controller ready?
8BB5   A9 00       LDA #$00        Write byte val,2nd part of space2
8BB7   8D 03 20    STA $2003       to diskette
8BBA   CA          DEX             Write next byte
8BBB   D0 EE       BNE $8BAB       All bytes done?
8BBD   A2 03       LDX #$03        Set counter
8BBF²  AD 00 20    LDA $2000       Get status register & isolate
8BC2   29 03       AND #$03        command bits
```

```
8BC4   4A          LSR A        Test bit for 'Busy'
8BC5   90 24       BCC $8BEB    Will command still be executed?
8BC7   F0 F6       BEQ $8BBF    YES—Data controller ready?
8BC9   A9 F5       LDA #$F5     Write value for time byte $A1
8BCB   8D 03 20    STA $2003    to diskette
8BCE   CA          DEX          Write next byte
8BCF   D0 EE       BNE $8BBF    All bytes done?
8BD1¹  AD 00 20    LDA $2000    Get status register & isolate
8BD4   29 03       AND #$03     command bits
8BD6   4A          LSR A        Test bit for 'Busy'
8BD7   90 12       BCC $8BEB    Will command still be executed?
8BD9   F0 F6       BEQ $8BD1    YES—Data controller ready?
8BDB   A9 FB       LDA #$FB     Write byte val:'Data Address
8BDD   8D 03 20    STA $2003    Save' to diskette
8BE0   84 6F       STY $6F      Save current sector pointer
8BE2   A4 44       LDY $44      Get number of sector portions
8BE4   EA          NOP          Two-cycle delay
8BE5³  AD 00 20    LDA $2000    Get status register & isolate
8BE8   29 03       AND #$03     command bits
8BEA   4A          LSR A        Test bit for 'Busy'
8BEB³  90 60       BCC $8C4D    Will command still be executed?
8BED   F0 F6       BEQ $8BE5    YES—Data controller ready?
8BEF   AD 0A 02    LDA $020A    Write empty byte for sector
8BF2   8D 03 20    STA $2003    to diskette
8BF5   EC 71 02    CPX $0271    Test for length of a sub-sector
8BF8   F0 04       BEQ $8BFE    Entire sub-sector written?
8BFA   E8          INX          NO—Write further to
8BFB   4C E5 8B    JMP $8BE5    next byte
8BFE¹  E8          INX          Initialize cntr:subsector length
8BFF   88          DEY          Decrement number of sub-sectors
8C00   D0 E3       BNE $8BE5    Write to other sub-sectors?
8C02¹  AD 00 20    LDA $2000    NO—Get status register & isolate
8C05   29 03       AND #$03     command bits
8C07   4A          LSR A        Test bit for 'Busy'
8C08   90 43       BCC $8C4D    Will command still be executed?
8C0A   F0 F6       BEQ $8C02    YES—Data controller ready?
8C0C   A9 F7       LDA #$F7     Write byte value for 2 CRC-bytes
8C0E   8D 03 20    STA $2003    to diskette
8C11   AC 05 02    LDY $0205    Identifier for sector length
8C14   B9 4F 8C    LDA $8C4F,Y  Get size/spaces between sectors
8C17   A4 6F       LDY $6F      Number of current sector
8C19   AA          TAX          Set space counter
8C1A²  AD 00 20    LDA $2000    Get status register & isolate
8C1D   29 03       AND #$03     command bits
8C1F   4A          LSR A        Test bit for 'Busy'
8C20   90 2B       BCC $8C4D    Will command still be executed?
8C22   F0 F6       BEQ $8C1A    YES—Data controller ready?
```

```
8C24   A9 4E      LDA #$4E       Write byte value for space 3
8C26   8D 03 20   STA $2003      to diskette
8C29   CA         DEX            Write next byte
8C2A   D0 EE      BNE $8C1A      All bytes done?
8C2C   CC 07 02   CPY $0207      Number of sectors to track
8C2F   F0 04      BEQ $8C35      All sectors set up?
8C31   C8         INY            NO—Increment sector counter
8C32   4C 07 8B   JMP $8B07      write next sector
8C35³  AD 00 20   LDA $2000      Get status register & isolate
8C38   29 03      AND #$03       command bits
8C3A   4A         LSR A          Test bit for 'Busy'
8C3B   90 0B      BCC $8C48      Will command still be executed?
8C3D   F0 F6      BEQ $8C35      YES—Data controller ready?
8C3F   18         CLC            Write byte value
8C40   A9 4E      LDA #$4E       for space 4
8C42   8D 03 20   STA $2003      to diskette
8C45   4C 35 8C   JMP $8C35      Fill rest of track
8C48¹  20 61 88   JSR $8861      Wait until command is finished
8C4B   18         CLC            Set flag for 'formatting  Ok'
8C4C   24         .byte $24      Jump to next byte (bit command)
8C4D³  38         SEC            Set flag for format error
8C4E   60         RTS            Return from this subroutine
---------------------------------------------------------------------
[8C14]  Number of bytes for spaces between CP/M sectors
8C4F   07                        Value for  128 bytes per sector
8C50   0C                        Value for  256 bytes per sector
8C51   17                        Value for  512 bytes per sector
8C52   2C                        Value for 1024 bytes per sector
---------------------------------------------------------------------
[8CA7]  Number of CP/M sectors per track by formatting
8C53   1A                        Value for  128 bytes per sector
8C54   10                        Value for  256 bytes per sector
8C55   09                        Value for  512 bytes per sector
8C56   05                        Value for 1024 bytes per sector
---------------------------------------------------------------------
[Vector: 86D8]
Format diskette in 'IBM System 34'
8C57   A5 3B      LDA #$3B       Test for
8C59   29 08      AND #$08       write-protect flag
8C5B   F0 07      BEQ $8C64      Is 'Write Protect' set?
8C5D   A6 46      LDX $46        YES—Get error number
8C5F   8E B0 01   STX $01B0      & set as return message
8C62   38         SEC            Flag for 'Error encountered'
8C63   60         RTS            Return from this subroutine
8C64¹  20 07 D3   JSR $D307      Clear all channels
8C67   AD 74 02   LDA $0274      Lengh of command string
8C6A   38         SEC            Draw off number of bytes utilized
```

```
8C6B   E9 04      SBC #$04      and
8C6D   A8         TAY           save value
8C6E   F0 20      BEQ $8C90     Any more statements onhand?
8C70   88         DEY           Pointer to next command byte
8C71   F0 22      BEQ $8C95     More statements in cmd string?
8C73   A9 00      LDA #$00      YES-Set first track to be
8C75   8D B0 01   STA $01B0     formatted
8C78   AD 05 02   LDA $0205     Get identifier for 2nd length &
8C7B   20 57 8A   JSR $8A57     set appropriate pointer
8C7E   88         DEY           Pointer to next command byte
8C7F   F0 21      BEQ $8CA2     More statements in cmd string?
8C81   88         DEY           YES-Pointer to next command byte
8C82   F0 23      BEQ $8CA7     More statements in cmd string?
8C84   88         DEY           YES-Pointer to next command byte
8C85   F0 26      BEQ $8CAD     More statements in cmd string?
8C87   88         DEY           YES-Pointer to next command byte
8C88   F0 2B      BEQ $8CB5     More statements in cmd string?
8C8A   88         DEY           YES-Pointer to next command byte
8C8B   F0 2D      BEQ $8CBA     More statements in cmd string?
8C8D   4C BF 8C   JMP $8CBF     Set no insert-value
8C90¹  A9 00      LDA #$00      Clear track statement
8C92   8D 04 02   STA $0204     in command string
8C95¹  A9 00      LDA #$00      Set first track to be
8C97   8D B0 01   STA $01B0     formatted (0)
8C9A   A9 01      LDA #$01      Set identifier for 256 bytes per
8C9C   8D 05 02   STA $0205     sector
8C9F   20 57 8A   JSR $8A57     Set sector pointer
8CA2¹  A9 27      LDA #$27      Lay out size of
8CA4   8D 06 02   STA $0206     formatted track
8CA7¹  BD 53 8C   LDA $8C53,X   Determine & set number
8CAA   8D 07 02   STA $0207     of sectors per track
8CAD¹  A9 00      LDA #$00      Set first logical
8CAF   8D 08 02   STA $0208     track number
8CB2   8D 01 20   STA $2001     Give track over CP/M controller
8CB5¹  A9 00      LDA #$00      Set first physical track to be
8CB7   8D 09 02   STA $0209     formatted
8CBA¹  A9 E5      LDA #$E5      Save empty bytes to fill
8CBC   8D 0A 02   STA $020A     sectors
8CBF¹  20 DE 8C   JSR $8CDE     Format disk side in IBM format
8CC2   AD B0 01   LDA $01B0     Get return message & compare
8CC5   E0 02      CPX #$02      with value for 'Ok'
8CC7   B0 12      BCS $8CDB     Is track formatting error-free?
8CC9   A5 3B      LDA $3B       YES-Get command number & test
8CCB   29 20      AND #$20      flag for 'two sides'
8CCD   F0 0C      BEQ $8CDB     Should both sides be formatted?
8CCF   A5 3B      LDA $3B       YES-Set flag for
8CD1   09 10      ORA #$10      side 2 in
```

```
8CD3   85 3B      STA $3B          command number
8CD5   20 54 89   JSR $8954        Activate head on current side
8CD8   20 DE 8C   JSR $8CDE        Format disk side in IBM format
8CDB2  4C EF 89   JMP $89EF        Set head to track 0 & end
--------------------------------------------------------------------

[8CBF/8CD8]
Format disk side in 'IBM System 34'
8CDE   20 B3 89   JSR $89B3        Initialize track
8CE1   B0 7C      BCS $8D5F        Is there an index hole?
8CE3   A9 01      LDA #$01         YES—Re-initialize flag for disk
8CE5   8D 0D 18   STA $180D        exchange (write-protect changed)
8CE8   20 EF 89   JSR $89EF        Set head for track 0
8CEB   AD 08 02   LDA $0208        Get 9th char from command string
8CEE   8D B0 01   STA $01B0        & set as first track number
8CF1   8D 01 20   STA $2001        Send track # over CP/M controller
8CF4   2C 03 02   BIT $0203        Get 4th char from command string
8CF7   70 05      BVS $8CFE        Flag for 'no sector table' set?
8CF9   20 88 88   JSR $8888        NO—Create sector table
8CFC   B0 61      BCS $8D5F        Table created withou errors?
8CFE1  AD 09 02   LDA $0209        YES—Get 10th char from cmndstring
8D01   29 7F      AND #$7F         Set as first physical track #
8D03   F0 08      BEQ $8D0D        Head moved to a starting track?
8D05   18         CLC              YES—Physical track at start-of-
8D06   65 67      ADC $67          format should be
8D08   85 67      STA $67          computed
8D0A   20 BA 87   JSR $87BA        Control track
8D0D2  78         SEI              Disable bus/controller interrupt
8D0E   AD 0D 18   LDA $180D        Test signal from circuitry for
8D11   4A         LSR A            'Write-protect has to be changed'
8D12   B0 4B      BCS $8D5F        Has diskette been changed?
8D14   20 86 8A   JSR $8A86        NO—Format track
8D17   B0 46      BCS $8D5F        Has an error been found?
8D19   AD 0D 18   LDA $180D        NO—Test signal from circuitry for
8D1C   4A         LSR A            'Write-protect has to be changed'
8D1D   B0 40      BCS $8D5F        Diskette been changed?
8D1F   20 F0 88   JSR $88F0        NO—Test sectors
8D22   B0 3B      BCS $8D5F        All sectors written error-free?
8D24   AD 0D 18   LDA $180D        YES—Test signal frm circuitry for
8D27   4A         LSR A            'Write-protect has to be changed'
8D28   B0 35      BCS $8D5F        Has diskette been changed?
8D2A   AD B0 01   LDA $01B0        Compare current logical track #
8D2D   CD 06 02   CMP $0206        with last number
8D30   F0 0E      BEQ $8D40        Is desired range formatted?
8D32   E6 67      INC $67          NO—Set to next target track
8D34   EE 01 20   INC $2001        Put CP/M controller to next track
8D37   EE B0 01   INC $01B0        Increment current track number
8D3A   20 BA 87   JSR $87BA        Set head to track
```

```
8D3D   4C 0D 8D   JMP $8D0D       Keep formatting
8D40¹  24 3B      BIT $3B         Test flag in command number
8D42   10 18      BPL $8D5C       End track recognized / cleared?
8D44   38         SEC             Number of tracks to be formatted
8D45   AD 06 02   LDA $0206       from last logical track
8D48   ED 08 02   SBC $0208       & compute first track number
8D4B   C9 27      CMP #$27        Compare with maximum # of tracks
8D4D   B0 0D      BCS $8D5C       Everything til side 2 formatted?
8D4F   E6 67      INC $67         YES—Go to side 2 for
8D51   20 BA 87   JSR $87BA       formatting
8D54   A2 1C      LDX #$1C        Write 7168 times $55 (%01010101)
8D56   20 63 9D   JSR $9D63       to track
8D59   20 00 FE   JSR $FE00       Switch head to read
8D5C²  A2 00      LDX #$00        Error number for 'Ok'
8D5E   2C         .byte $2C       Jump to next 2 bytes(bit command)
8D5F⁶  A2 06      LDX #$06        Error number for 'Format error'
8D61   8E B0 01   STX $01B0       Set return message &
8D64   4C E9 85   JMP $85E9       get ready for output
--------------------------------------------------------------------
[8DF0/Vektor: 86D9]
Read CP/M sector & send to computer
8D67   A5 3B      LDA $3B         Get command #, & test Flag for
8D69   29 20      AND #$20        'Buffer output only'
8D6B   D0 59      BNE $8DC6       Set?
8D6D   A9 03      LDA #$03        NO—Set current buffer pointer to
8D6F   85 31      STA $31         starting address
8D71   A0 00      LDY #$00        from buffer 0
8D73   84 30      STY $30         ($0300)
8D75   A6 44      LDX $44         Get number of sub-sectors
8D77   AD 03 02   LDA $0203       Give track number to
8D7A   8D 01 20   STA $2001       CP/M controller
8D7D   AD 04 02   LDA $0204       Give number of desired sector
8D80   8D 02 20   STA $2002       over CP/M controller
8D83   A9 88      LDA #$88        %10001000 'Read sector'
8D85   20 4E 88   JSR $884E       command  over CP/M controller
8D88   EA         NOP             Two-cycle delay
8D89³  AD 00 20   LDA $2000       Get status register & isolate
8D8C   29 03      AND #$03        command bits
8D8E   4A         LSR A           Test bit for 'Busy'
8D8F   90 1A      BCC $8DAB       Will command still be executed?
8D91   29 01      AND #$01        YES—Flagbit:'Data register ready'
8D93   F0 F4      BEQ $8D89       Wait until data are ready
8D95   AD 03 20   LDA $2003       Get Data byte from CP/M
8D98   91 30      STA ($30),Y     controller & write in buffer
8D9A   CC 71 02   CPY $0271       Number of bytes per sub-sector
8D9D   F0 03      BEQ $8DA2       All bytes read?
8D9F   C8         INY             NO—Buffer pointer to next byte
```

```
8DA0    D0 E7       BNE $8D89       End of buffer reached?
8DA2[1] C8          INY             YES—Clear buffer pointer
8DA3    CA          DEX             Next sub-sector
8DA4    F0 05       BEQ $8DAB       All sub-sectors read?
8DA6    E6 31       INC $31         NO—Buffer pointer to next buffer
8DA8    4C 89 8D    JMP $8D89       Keep reading sectors
8DAB[2] 20 61 88    JSR $8861       Wait until command is done
8DAE    20 3C 88    JSR $883C       Get status of CP/M controller
8DB1    20 E9 85    JSR $85E9       Prepare error number for output
8DB4    24 3B       BIT $3B         Test flag for 'Error noted'
8DB6    70 07       BVS $8DBF       Should return message be tested?
8DB8    E0 02       CPX #$02        YES—Test against value for 'OK'
8DBA    90 03       BCC $8DBF       Is number greater (error number)?
8DBC    4C 84 83    JMP $8384       YES—Display error
8DBF[2] 20 F9 85    JSR $85F9       Output byte over 1571 bus
8DC2    A5 3B       LDA $3B         Note flag for 'read sector only'
8DC4    30 22       BMI $8DE8       Should buffer be transferred?
8DC6[1] A9 03       LDA #$03        YES—Set current buffer pointers
8DC8    85 31       STA $31         ($30/$31) to starting address
8DCA    A0 00       LDY #$00        in ($0300)
8DCC    84 30       STY $30         buffer 0 ($0300)
8DCE    A6 44       LDX $44         Number of sub-sectors per sector
8DD0[2] B1 30       LDA ($30),Y     Get byte from buffer & save
8DD2    85 46       STA $46         as character to be output
8DD4    20 F9 85    JSR $85F9       Output byte over 1571 bus
8DD7    CC 71 02    CPY $0271       Number of bytes per sub-sector
8DDA    F0 03       BEQ $8DDF       Entire sub-sector already sent?
8DDC    C8          INY             NO—Buffer pointer to next byte
8DDD    D0 F1       BNE $8DD0       Reached end-of-buffer?
8DDF[1] C8          INY             YES—Set buffer pointer to start
8DE0    CA          DEX             Decrement number of sub-sectors
8DE1    F0 05       BEQ $8DE8       Whole sector already been sent?
8DE3    E6 31       INC $31         NO—Buffer pointer to next buffer
8DE5    4C D0 8D    JMP $8DD0       Continue sending data
8DE8[2] CE 05 02    DEC $0205       # of sectors to be transferred
8DEB    F0 06       BEQ $8DF3       Read more sectors?
8DED    20 6C 88    JSR $886C       YES—Compute next sector number
8DF0    4C 67 8D    JMP $8D67       Read next sector
8DF3[1] 4C 1B 89    JMP $891B       Control next given track
------------------------------------------------------------------
```

```
[8EBF/Vector: 86DC]
Send CP/M sector from computer and write to diskette
8DF6  A9 03     LDA #$03     Set curr. buffer pointer $30/$31
8DF8  85 31     STA $31      to starting address
8DFA  A0 00     LDY #$00     from buffer
8DFC  84 30     STY $30      0 ($0300)
8DFE  A6 44     LDX $44      Number of sections per sector
8E00  A5 3B     LDA $3B      Test flag for 'buffer read'
8E02  30 30     BMI $8E34    Data taken from computer?
8E04² AD 00 18  LDA $1800    YES—Get bus control register &
8E07  49 08     EOR #$08     switch to Clock output
8E09  2C 0D 40  BIT $400D    Re-set interrupt register
8E0C  8D 00 18  STA $1800    Set bus control register
8E0F¹ AD 00 18  LDA $1800    Test ATN input
8E12  10 03     BPL $8E17    Set?
8E14  20 59 EA  JSR $EA59    Test for ATN command mode
8E17¹ AD 0D 40  LDA $400D    NO—Test flag for 'serial
8E1A  29 08     AND #$08     input register full'
8E1C  F0 F1     BEQ $8E0F    Is data transmitting?
8E1E  AD 0C 40  LDA $400C    YES—Get byte and write
8E21  91 30     STA ($30),Y  to buffer
8E23  CC 71 02  CPY $0271    Number of bytes per sub-sector
8E26  F0 03     BEQ $8E2B    Entire subdivision read in?
8E28  C8        INY          NO—Buffer pointer to next byte
8E29  D0 D9     BNE $8E04    End of buffer reached?
8E2B¹ C8        INY          YES—Set buffer pointer to start
8E2C  CA        DEX          Next sub-sector
8E2D  F0 05     BEQ $8E34    Read more subdivisions from bus?
8E2F  E6 31     INC $31      YES—Buffer pointer to next buffer
8E31  4C 04 8E  JMP $8E04    Continue reading
8E34¹ A5 3B     LDA $3B      Get command number and flag for
8E36  29 20     AND #$20     'Write buffer in sector'
8E38  D0 7D     BNE $8EB7    Should sector be written?
8E3A  A5 3B     LDA $3B      YES—Test out flag for
8E3C  29 08     AND #$08     'Write protect'
8E3E  F0 05     BEQ $8E45    Is write-protect active?
8E40  A6 46     LDX $46      YES—Get error number and
8E42  4C 81 83  JMP $8381    output it
8E45¹ A9 03     LDA #$03     Set curr. buffer pointer $30/$31
8E47  85 31     STA $31      to starting address
8E49  A0 00     LDY #$00     from buffer
8E4B  84 30     STY $30      0 ($0300)
8E4D  A6 44     LDX $44      Number of subsectors per sector
8E4F  AD 03 02  LDA $0203    Get  track # from command string,
8E52  8D 01 20  STA $2001    and give to CP/M controller
8E55  AD 04 02  LDA $0204    Get number of desired sector and
8E58  8D 02 20  STA $2002    give to CP/M controller
```

```
8E5B    AD 0D 18   LDA $180D      Signal from circuitry for
8E5E    4A         LSR A          'Write-Protect has been changed'
8E5F    B0 32      BCS $8E93      Has diskette been changed?
8E61    A9 A8      LDA #$A8       NO—%101010000 Convey'Write single
8E63    20 4E 88   JSR $884E      sector'command to CP/M controller
8E66[3] AD 00 20   LDA $2000      Get status register and
8E69    29 03      AND #$03       isolate command bits
8E6B    4A         LSR A          Test bit for 'Busy'
8E6C    90 25      BCC $8E93      Is command yet to be executed?
8E6E    29 01      AND #$01       YES—Flag: 'data register empty'
8E70    F0 F4      BEQ $8E66      Should new data be taken up?
8E72    B1 30      LDA ($30),Y    YES—Get data byte from buffer and
8E74    8D 03 20   STA $2003      write to diskette
8E77    CC 71 02   CPY $0271      Number of bytes per subdivision
8E7A    F0 03      BEQ $8E7F      End of sub-sectors?
8E7C    C8         INY            NO—buffer pointer to next byte
8E7D    D0 E7      BNE $8E66      End of buffer reached?
8E7F[1] C8         INY            YES—Set buffer pointer to start
8E80    CA         DEX            Number of subdivisions per sector
8E81    F0 05      BEQ $8E88      Still another sub-sector?
8E83    E6 31      INC $31        YES—Buffer address to next buffer
8E85    4C 66 8E   JMP $8E66      Keep writing to diskette
8E88    AD 0D 18   LDA $180D      Check signal from circuitry for
8E8B    4A         LSR A          'Write-protect has to be changed'
8E8C    B0 05      BCS $8E93      Has diskette been exchanged?
8E8E    20 C6 8E   JSR $8EC6      NO—Read sector to test
8E91    90 07      BCC $8E9A      Read functions perfectly?
8E93[3] 20 CE 81   JSR $81CE      NO—Switch 1571 bus to output
8E96    A2 07      LDX #$07       Error number for 'verify error'
8E98    D0 06      BNE $8EA0      Jump to $8EA0
8E9A[1] 20 CE 81   JSR $81CE      Switch 1571 bus to output
8E9D    20 3C 88   JSR $883C      Get CP/M controller error status
8EA0[1] 8E B0 01   STX $01B0      and save it
8EA3    20 E9 85   JSR $85E9      Prepare error fo output
8EA6    20 F9 85   JSR $85F9      Send byte over 1571 bus
8EA9    20 A0 86   JSR $86A0      Wait for jumper from Clock
8EAC    20 B2 81   JSR $81B2      Switch 1571 bus to input
8EAF    24 3B      BIT $3B        Test 'Note error' flag
8EB1    70 04      BVS $8EB7      Return message to be verified?
8EB3    E0 02      CPX #$02       YES—Verify against error number
8EB5    B0 0E      BCS $8EC5      Is there an error?
8EB7[2] CE 05 02   DEC $0205      NO—# of sectors to be written
8EBA    F0 06      BEQ $8EC2      Any more sectors?
8EBC    20 6C 88   JSR $886C      YES—Get number of next sector
8EBF    4C F6 8D   JMP $8DF6      Read and write next sector
8EC2[1] 4C 1B 89   JMP $891B      Get next track and set it
8EC5[1] 60         RTS            Return from this subroutine
```

```
[8E8E/Vector: 86DE]
Compare CP/M sector with buffer contents (verify)
8EC6    A9 03       LDA #$03       Set curr. buffer pointer $30/$31
8EC8    85 31       STA $31        to starting
8ECA    A0 00       LDY #$00       address from
8ECC    84 30       STY $30        buffer 0 ($0300)
8ECE    A6 44       LDX $44        Number of subsectors per sector
8ED0    AD 03 02    LDA $0203      Get track # from command string &
8ED3    8D 01 20    STA $2001      send to CP/M controller
8ED6    AD 04 02    LDA $0204      Get number of desired sector and
8ED9    8D 02 20    STA $2002      send to CP/M controller
8EDC    A9 88       LDA #$88       %10001000 'Read Sector'
8EDE    20 4E 88    JSR $884E      Send command to controller
8EE1[3] AD 00 20    LDA $2000      Get status register and
8EE4    29 03       AND #$03       isolate command bits
8EE6    4A          LSR A          Test bit for 'Busy'
8EE7    90 1C       BCC $8F05      Command yet to be executed?
8EE9    29 01       AND #$01       YES—Test 'Ready for data' flag
8EEB    F0 F4       BEQ $8EE1      Wait until data byte is ready?
8EED    AD 03 20    LDA $2003      Read byte from diskette
8EF0    D1 30       CMP ($30),Y    and compare with buffer contents
8EF2    D0 11       BNE $8F05      Identical?
8EF4    CC 71 02    CPY $0271      YES—Number of bytes/sub-sector
8EF7    F0 03       BEQ $8EFC      Entire subdivision compared?
8EF9    C8          INY            NO—buffer pointer to next byte
8EFA    D0 E5       BNE $8EE1      End of buffer reached?
8EFC[1] C8          INY            YES—Set buffer pointer to start
8EFD    CA          DEX            Number of sub-sectors
8EFE    F0 10       BEQ $8F10      Any sub-sectors left?
8F00    E6 31       INC $31        YES—Pointer addr to next buffer
8F02    4C E1 8E    JMP $8EE1      Continue verify
8F05[2] A9 D0       LDA #$D0       %11010000 'Forced Interrupt'
8F07    8D 00 20    STA $2000      on controller; verify ends
8F0A    20 83 A4    JSR $A483      Approx. 80-cycle delay
8F0D    A2 07       LDX #$07       Error number for 'verify error'
8F0F    2C          .byte $2C      Jump to next 2 bytes(bit command)
8F10[1] A2 00       LDX #$00       Error number for 'Ok'
8F12    8E B0 01    STX $01B0      Save number
8F15    4C 61 88    JMP $8861      Wait for end of command
---------------------------------------------------------------------
```

```
[8900/Vector: 86E0]
Test CP/M sector for empty contents
8F18  A9 03     LDA #$03     Set curr. buffer pointer $30/$31
8F1A  85 31     STA $31      to starting address
8F1C  A0 00     LDY #$00     from
8F1E  84 30     STY $30      buffer 0 ($0300)
8F20  A6 44     LDX $44      Number of sub-sectors per sector
8F22  AC 71 02  LDY $0271    Length of a sub-sector
8F25  A9 88     LDA #$88     %10001000 'Read Sector'
8F27  20 4E 88  JSR $884E    Give command to CP/M controller
8F2A3 AD 00 20  LDA $2000    Get status register and
8F2D  29 03     AND #$03     isolate command bits
8F2F  4A        LSR A        Test bit for 'Busy'
8F30  90 1A     BCC $8F4C    Command still need to be run?
8F32  29 01     AND #$01     YES—Test 'Ready for data' flag
8F34  F0 F4     BEQ $8F2A    Waiting for a data byte?
8F36  AD 03 20  LDA $2003    Read byte from diskette & compare
8F39  CD 0A 02  CMP $020A    with value for empty byte
8F3C  D0 0E     BNE $8F4C    Identical?
8F3E  88        DEY          YES—Next byte
8F3F  10 E9     BPL $8F2A    Entire sub-sector compared?
8F41  CA        DEX          YES—Number of sub-sectors
8F42  F0 13     BEQ $8F57    Any more sub-sectors?
8F44  AC 71 02  LDY $0271    YES—Set counter again
8F47  E6 31     INC $31      Buffer pointer to next buffer
8F49  4C 2A 8F  JMP $8F2A    Continue testing
8F4C2 A9 D0     LDA #$D0     '11010000' 'Forced Interrupt' to
8F4E  8D 00 20  STA $2000    controller; command interrupt
8F51  20 83 A4  JSR $A483    Wait approx. 80 cycles
8F54  A2 07     LDX #$07     Error number for 'verify error'
8F56  2C        .byte $2C    Jump to next two bytes
8F571 A2 00     LDX #$02     Error # for 'Header not found'
8F59  8E B0 01  STX $01B0    Set number
8F5C  4C 61 88  JMP $8861    Wait until command is finished
-----------------------------------------------------------------
[Vector: 86E2]
Read all CP/M headers and determine sector sequence
8F5F  08        PHP          Retain processor status
8F60  78        SEI          Disable bus/controller interrupt
8F61  20 EF 89  JSR $89EF    Set head to track 0
8F64  24 3B     BIT $3B      Test 'Track set' flag
8F66  10 08     BPL $8F70    Should a new track be turned to?
8F68  AD 03 02  LDA $0203    YES—Get track # from cmd string
8F6B  85 67     STA $67      and set as target track
8F6D  20 BA 87  JSR $87BA    Position head to track
8F701 A9 00     LDA #$00     Clear counter for number
8F72  85 97     STA $97      of sectors
```

```
8F74   20 27 8A   JSR $8A27        Read next header
8F77   AE B0 01   LDX $01B0        Get return message and check
8F7A   E0 02      CPX #$02         against error number
8F7C   B0 1F      BCS $8F9D        Read procedure done, error-free?
8F7E   A5 26      LDA $26          YES—Get sector number and save
8F80   85 96      STA $96          as first sector number
8F82¹  20 27 8A   JSR $8A27        Read next header
8F85   A5 26      LDA $26          Get sector number
8F87   A4 97      LDY $97          Pointer to curr. sector position
8F89   99 0B 02   STA $020B,Y      Enter sector number in table
8F8C   E6 97      INC $97          Pointer to next sector entry
8F8E   C0 1F      CPY #$1F         Compare with max. # of sectors
8F90   B0 0B      BCS $8F9D        Number of sectors allowable?
8F92   C5 96      CMP $96          YES—Test against first sector #
8F94   D0 EC      BNE $8F82        Reached the first sector again?
8F96   A5 24      LDA $24          YES—Get track # from header &
8F98   85 67      STA $67          set as current target track
8F9A   A2 00      LDX #$00         Value for 'Ok' message
8F9C   2C         .byte $2C        Jump to next 2 bytes(bit command)
8F9D²  A2 02      LDX #$02         Error number: 'Header not found'
8F9F   8E B0 01   STX $01B0        Set return message
8FA2   28         PLP              Re-establish processor status
8FA3   60         RTS              Return from this subroutine
--------------------------------------------------------------------
[8FF1]
'S'-command (sector) : Set sector format for Commodore diskettes
8FA4   AD 04 02   LDA $0204        Get 5th char from command string
8FA7   85 69      STA $69          and set as new sector format
8FA9   60         RTS              Return from this subroutine
--------------------------------------------------------------------
[8FF5]
'R'-command (Read) : Set number of read attempts
8FAA   AD 04 02   LDA $0204        Get 5th char from command string
8FAD   85 6A      STA $6A          and set as new # of read attempts
8FAF   60         RTS              Return from this subroutine
--------------------------------------------------------------------
[8FF9]
'T'-command (Test) : Test ROM checksum
8FB0   4C 4E 92   JMP $924E        Compute checksum
--------------------------------------------------------------------
[9001]
'H'-command (Head) : Set head at given diskette side
(in 1541 mode only)
8FB3   78         SEI              Disable bus/controller interrupt
8FB4   AD 0F 18   LDA $180F        Get control register and get
8FB7   29 20      AND #$20         flag for operating mode
8FB9   D0 66      BNE $9021        Is drive in 1541 mode?
```

```
8FBB  AD 04 02    LDA $0204       YES—Get 5th char from cmnd string
8FBE  C9 31       CMP #$31        and compare with '1'
8FC0  F0 12       BEQ $8FD4       Should head be set to side 2?
8FC2  C9 30       CMP #$30        NO—Compare with '0'
8FC4  D0 5B       BNE $9021       Should head be set to side 1?
8FC6  AD 0F 18    LDA $180F       YES—Get control register and
8FC9  29 FB       AND #$FB        switch head circuitry to
8FCB  8D 0F 18    STA $180F       side 1
8FCE  58          CLI             Enable bus/controller interrupt
8FCF  24 3B       BIT $3B         Test flag in command number
8FD1  10 0E       BPL $8FE1       Should diskette be initialized?
8FD3  60          RTS             NO—Return from this subroutine
------------------------------------------------------------
[8FC0]
Change head to side 2
8FD4  AD 0F 18    LDA $180F       Get control register and
8FD7  09 04       ORA #$04        place bit for head electronics to
8FD9  8D 0F 18    STA $180F       side 2
8FDC  58          CLI             Enable bus/controller interrupt
8FDD  24 3B       BIT $3B         Test flag in command number
8FDF  30 03       BMI $8FE4       Should diskette be initialized?
8FE1¹ 4C 42 D0    JMP $D042       YES—Read BAM from diskette
8FE4¹ 60          RTS             Return from this subroutine
------------------------------------------------------------
[Origin of vector 80C2 through routine 8030]
Decode status/control functions
8FE5  AE 74 02    LDX $0274       Determine length/command string
8FE8  E0 04       CPX #$04        and test if 4 chars are given
8FEA  90 35       BCC $9021       Command a minimum 4 char long?
8FEC  AD 03 02    LDA $0203       YES—Get 4th char from command
8FEF  C9 53       CMP #$53        and compare with 'S'
8FF1  F0 B1       BEQ $8FA4       Should sector format be set?
8FF3  C9 52       CMP #$52        NO—Compare with 'R'
8FF5  F0 B3       BEQ $8FAA       Set number of read attempts?
8FF7  C9 54       CMP #$54        NO—Compare with 'T'
8FF9  F0 B5       BEQ $8FB0       Test ROM-checksum?
8FFB  C9 4D       CMP #$4D        NO—Compare with 'M'
8FFD  F0 27       BEQ $9026       1541/1571 mode switched around?
8FFF  C9 48       CMP #$48        NO—Compare with 'H'
9001  F0 B0       BEQ $8FB3       Should diskette side be changed?
------------------------------------------------------------
```

```
Set device address (number in A)
9003  A8          TAY               Save device address
9004  C0 04       CPY #$04          Compare with minimal IEC address
9006  90 19       BCC $9021         Is new address smaller?
9008  C0 1F       CPY #$1F          NO—Check maximum IEC address
900A  B0 15       BCS $9021         New address in allowable range?
900C  A9 40       LDA #$40          YES—Set identifier for
900E  85 78       STA $78           Talk
9010  A9 20       LDA #$20          Set identifier for
9012  85 77       STA $77           Listen
9014  98          TYA               Get new device address and use it
9015  18          CLC               to set new address for
9016  65 78       ADC $78           Talk call
9018  85 78       STA $78           Set address
901A  98          TYA               Get new device address and use it
901B  18          CLC               to establish new address for
901C  65 77       ADC $77           Listen call
901E  85 77       STA $77           Set address
9020  60          RTS               Return from this subroutine
------------------------------------------------------------
[8FB9/8FC4/8FEA/9006/900A/9030]
9021  A9 31       LDA #$31          Display
9023  4C C8 C1    JMP $C1C8         '31 Syntax Error' message
------------------------------------------------------------
[8FFD]
'M'-command (Mode) : 1541 / 1571 operating mode switching
9026  78          SEI               Disable bus/controller interrupt
9027  AD 04 02    LDA $0204         Get 5th char from command string
902A  C9 31       CMP #$31          and compare with '1'
902C  F0 20       BEQ $904E         Switched into 1571 mode?
902E  C9 30       CMP #$30          NO—Compare with '0'
9030  D0 EF       BNE $9021         Switched into 1541 mode?
------------------------------------------------------------
Switch to 1541 mode
9032  AD 0F 18    LDA $180F         YES—Get control register
9035  29 DF       AND #$DF          Switch control & bus electronics
9037  8D 0F 18    STA $180F         to 1541 (1 MHz speed)
903A  20 83 A4    JSR $A483         80-cycle delay
903D  20 82 FF    JSR $FF82         Initialize 1541 mode
9040  AD AF 02    LDA $02AF         Set flag for
9043  09 80       ORA #$80          '1541 IRQ Routine'
9045  8D AF 02    STA $02AF         ($9D88)
9048  58          CLI               Enable bus/controller interrupt
9049  24 3B       BIT $3B           Test flag in command number
904B  10 2F       BPL $907C         Should diskette be initialized?
904D  60          RTS               NO—Return from this subroutine
------------------------------------------------------------
```

```
[902C]
Switch to 1571 mode
904E  AD 0F 18   LDA $180F      Get control register and
9051  09 20      ORA #$20       switch bus, operating electronics
9053  8D 0F 18   STA $180F      to 1571 (2 MHz speed)
9056  20 83 A4   JSR $A483      80-cycle delay
9059  A9 DE      LDA #$DE       IRQ vector in $02A9/$02AA to
905B  8D A9 02   STA $02A9      job loop call
905E  A9 9D      LDA #$9D       Turn to 1571 routine in
9060  8D AA 02   STA $02AA      $9DDE
9063  A9 40      LDA #$40       Timer 1 (High-Byte)
9065  8D 07 1C   STA $1C07      set to about
9068  8D 05 1C   STA $1C05      8 ms
906B  AD AF 02   LDA $02AF      Set flag for
906E  29 7F      AND #$7F       'Toggle IRQ from 1541 to
9070  8D AF 02   STA $02AF      1571'
9073  A9 00      LDA #$00       Clear flag for current
9075  85 62      STA $62        headmode
9077  58         CLI            Enable bus/controller interrupt
9078  24 3B      BIT $3B        Test command number
907A  30 03      BMI $907F      Should diskette be initialized?
907C1 4C 42 D0   JMP $D042      YES—Read BAM from diskette
907F1 60         RTS            Return from this subroutine
-----------------------------------------------------------------
[BF66/Origin over vector in 80CC through routine 8030]
Fast-load file over 1571 bus (PRG, SEQ or USR)
9080  20 CE 81   JSR $81CE      Switch 1571 bus to output
9083  20 EA 91   JSR $91EA      Prepare filename
9086  B0 5F      BCS $90E7      Improper filename?
9088  20 3D C6   JSR $C63D      NO—Initialize diskette
908B  A5 FF      LDA $FF        Get flag for drive status
908D  D0 58      BNE $90E7      Is drive ready?
908F  A5 37      LDA $37        YES—Get bus status and set
9091  09 81      ORA #$81       flags for '1571 mode' and
9093  85 37      STA $37        'last sector'
9095  20 CA 91   JSR $91CA      Set channel and buffer parameters
9098  AD 00 02   LDA $0200      Get first character of filename &
909B  C9 2A      CMP #$2A       compare with wildcard '*'
909D  D0 0F      BNE $90AE      Load last-loaded file?
909F  A5 7E      LDA $7E        YES—Get number of last track
90A1  F0 0B      BEQ $90AE      Is track number given?
90A3  48         PHA            YES—Save number
90A4  AD 6F 02   LDA $026F      Get number of last sector & enter
90A7  8D 85 02   STA $0285      in table
90AA  68         PLA            Get last track number
90AB  4C EC 90   JMP $90EC      Load file
90AE2 A9 00      LDA #$00       Clear pointer and register:
```

```
90B0  A8        TAY            [Error-- see 7.1.5]
90B1  AA        TAX            [Unnecessary initialization]
90B2  8D 8E 02  STA $028E      Number of last drive
90B5  8D 7A 02  STA $027A      Pointer to first filename
90B8  20 12 C3  JSR $C312      Get drive # from command string
90BB  AD 78 02  LDA $0278      Retain number of filenames
90BE  48        PHA            found and allow
90BF  A9 01     LDA #$01       for only one
90C1  8D 78 02  STA $0278      filename
90C4  A9 FF     LDA #$FF       Clear pointer in
90C6  85 86     STA $86        directory buffer
90C8  20 4F C4  JSR $C44F      Search for entry in directory
90CB  68        PLA            Repeat pointer with number
90CC  8D 78 02  STA $0278      of filenames
90CF  A5 37     LDA $37        Get bus status and
90D1  29 7F     AND #$7F       clear flag for
90D3  85 37     STA $37        '1571 mode'
90D5  24 3B     BIT $3B        Get command number and test flag
90D7  30 06     BMI $90DF      Should file be tested for 'PRG'?
90D9  A5 E7     LDA $E7        Determine filetype of file entry
90DB  C9 02     CMP #$02       & compare with identifier for PRG
90DD  D0 05     BNE $90E4      Is entry a PRG file?
90DF1 AD 80 02  LDA $0280      YES—Track # of first file sector
90E2  D0 08     BNE $90EC      Entry to be found in directory?
90E41 A2 02     LDX #$02       Error number for 'File Not Found'
90E6  2C        .byte $2C      Jump to next 2 bytes(bit command)
90E72 A2 0F     LDX #$0F       Error # for 'Drive Not Ready'
90E9  4C AD 91  JMP $91AD      Send error over 1571 bus
90EC2 85 7E     STA $7E        Save last track and
90EE  48        PHA            retain number
90EF  20 DA 91  JSR $91DA      Compute pointer to job table
90F2  68        PLA            Repeat last track number
90F3  AE B0 02  LDX $02B0      Get pointer from job table and
90F6  95 06     STA $06,X      set track of job
90F8  AD 85 02  LDA $0285      Get last sector number
90FB  8D 6F 02  STA $026F      and save down
90FE  95 07     STA $07,X      Give sector number to jobloop
9100  A9 80     LDA #$80       Read jobcode for
9102  8D 02 02  STA $0202      sector and save
9105  85 5F     STA $5F        as current jobcode
91071 58        CLI            Enable bus/controller interrupt
9108  A6 F9     LDX $F9        Get number of current buffer and
910A  A5 5F     LDA $5F        give current jobcode
910C  95 00     STA $00,X      to job loop
910E  20 4B 86  JSR $864B      Execute job
9111  E0 02     CPX #$02       Check return message to 'OK'
9113  90 03     BCC $9118      Job run error-free?
```

```
9115   4C 99 91   JMP $9199      NO—Display return message
9118¹  78         SEI            Disable bus/controller interrupt
9119   A0 00      LDY #$00       Buffer pnter to 1st byte/sector
911B   B1 94      LDA ($94),Y    Get byte from buffer
911D   F0 2F      BEQ $914E      Is this the last sector?
911F   A5 37      LDA $37        NO—Get bus status and
9121   29 FE      AND #$FE       clear flag for 'last
9123   85 37      STA $37        sector'
9125   20 28 92   JSR $9228      Give last 'OK'message by 1571 bus
9128   A0 02      LDY #$02       Buffer pointer to first data byte
912A¹  B1 94      LDA ($94),Y    Get byte from buffer and prepare
912C   AA         TAX            for output
912D   20 28 92   JSR $9228      Output byte over 1571 bus
9130   C8         INY            Turn buffer pointer to next byte
9131   D0 F7      BNE $912A      Entire buffer alredy transferred?
9133   AE B0 02   LDX $02B0      YES—Get pointer in job table
9136   B1 94      LDA ($94),Y    Track of next sector from buffer
9138   D5 06      CMP $06,X      Compare with track of last job
913A   F0 03      BEQ $913F      Next sector to same track?
913C   A0 80      LDY #$80       NO—Jobcode for 'Read sector'
913E   2C         .byte $2C      Jump to next 2 bytes(bit command)
913F¹  A0 88      LDY #$88       Jobcode'Read sector of sametrack'
9141   84 5F      STY $5F        Set jobcode and give
9143   95 06      STA $06,X      track number to job loop
9145   A0 01      LDY #$01       Pointer to number of next sector
9147   B1 94      LDA ($94),Y    Get byte from linked bytes and
9149   95 07      STA $07,X      give to job loop
914B   4C 07 91   JMP $9107      Transfer next sector
914E¹  A2 1F      LDX #$1F       Give return message for 'last
9150   20 28 92   JSR $9228      sector' over 1571 bus
9153   A9 01      LDA #$01       Test flag for 'only one sector'
9155   24 37      BIT $37        in bus status byte
9157   F0 1E      BEQ $9177      Does program have only one block?
9159   A8         TAY            YES—Set buffer pointer
915A   B1 94      LDA ($94),Y    Get # of data bytes applicable to
915C   38         SEC            sector and remove
915D   E9 03      SBC #$03       bytes for starting address &
915F   85 46      STA $46        linking bytes
9161   AA         TAX            Give # of bytes still to be
9162   20 28 92   JSR $9228      sent over 1571 bus
9165   C8         INY            Buffer pointer to prg start addr
9166   B1 94      LDA ($94),Y    Get lo-byte of start address &
9168   AA         TAX            set as character to be output
9169   20 28 92   JSR $9228      Send byte over 1571 bus
916C   C8         INY            Turn buffer pointer to hi-byte &
916D   B1 94      LDA ($94),Y    get byte from buffer
916F   AA         TAX            Give rest of starting address
```

```
9170   20 28 92   JSR $9228      over 1571 bus
9173   A0 04      LDY #$04       Set buffer pntr to begin. of data
9175   D0 0D      BNE $9184      Jump to $9184
9177¹  A0 01      LDY #$01       Pointer to data bytes yet allowed
9179   B1 94      LDA ($94),Y    Get # of data bytes from buffer
917B   AA         TAX            and save number
917C   CA         DEX            Send number of data bytes
917D   86 46      STX $46        still ahead over
917F   20 28 92   JSR $9228      1571 bus
9182   A0 02      LDY #$02       Pointer to start of data range
9184²  B1 94      LDA ($94),Y    Get byte from buffer and prepare
9186   AA         TAX            for output
9187   20 28 92   JSR $9228      Send byte over 1571 bus
918A   C8         INY            Turn buffer pointer to next byte
918B   C6 46      DEC $46        # of bytes yet to be transferred
918D   D0 F5      BNE $9184      All of them sent?
918F   A9 00      LDA #$00       YES—Set secondary address
9191   85 83      STA $83        for Load
9193   20 C0 DA   JSR $DAC0      Close file
9196   4C 94 C1   JMP $C194      Prepare return message
----------------------------------------------------------------
[9115/A9CF]
Display error message
9199   78         SEI            Disable bus/controller interrupt
919A   86 46      STX $46        Send error number over
919C   20 28 92   JSR $9228      1571 bus
919F   A9 00      LDA #$00       set secondary address for
91A1   85 83      STA $83        Load
91A3   20 C0 DA   JSR $DAC0      Close file
91A6   A6 F9      LDX $F9        Number of current buffer
91A8   A5 46      LDA $46        Error number
91AA   4C 0A E6   JMP $E60A      Prepare text version of message
----------------------------------------------------------------
[90E9]
Output Load error
91AD   78         SEI            Disable bus/controller interrupt
91AE   86 46      STX $46        Save error number
91B0   A2 02      LDX #$02       Error number for 'File Not Found'
91B2   20 28 92   JSR $9228      over 1571 bus
91B5   A9 00      LDA #$00       Set secondary address
91B7   85 83      STA $83        for load
91B9   20 C0 DA   JSR $DAC0      Close file
```

```
91BC  A5 46     LDA $46         Repeat error number and check
91BE  C9 02     CMP #$02        against 'File Not Found'
91C0  F0 03     BEQ $91C5       Identical?
91C2  A9 74     LDA #$74        NO—Number for 'Drive Not Ready'
91C4  2C A9 62  .byte $2C       Jump to next 2 bytes(bit command)
91C5¹ A9 62     LDA #$62        Number for 'File Not Found'
91C7  4C C8 C1  JMP $C1C8       Prepare text of message
--------------------------------------------------------------------

[9095]
Set up channel and buffer for Fast Load
91CA  A9 00     LDA #$00        Set secondary address for Load
91CC  85 83     STA $83         and set
91CE  A9 01     LDA #$01        number of buffer to be opened
91D0  20 E2 D1  JSR $D1E2       Open buffer and channel
91D3  AA        TAX             Get # of appropriate buffer and
91D4  BD E0 FE  LDA $FEE0,X     take high-byte of buffer address
91D7  85 95     STA $95         in buffer pointer
91D9  60        RTS             Return from this subroutine
--------------------------------------------------------------------

[90EF]
Find out track & sector numbers from job table
91DA  A5 95     LDA $95         Get high-byte of buffer pointer
91DC  38        SEC             & compute logical buffer # from
91DD  E9 03     SBC #$03        physical address; set
91DF  85 F9     STA $F9         as current buffer number
91E1  0A        ASL A           Double number (for 2-byte table)
91E2  8D B0 02  STA $02B0       and save it
91E5  A9 00     LDA #$00        Reset low-byte of buffer pointer
91E7  85 94     STA $94         to buffer start
91E9  60        RTS             Return from this subroutine
--------------------------------------------------------------------

[9083]
Shift filename to beginning of input buffer
91EA  A0 03     LDY #$03        Pointer to beginning of filename
91EC  AD 74 02  LDA $0274       Get length of command string &
91EF  38        SEC             take up character
91F0  E9 03     SBC #$03        for 'U0' command
91F2  8D 74 02  STA $0274       Save length of filename
91F5  AD 04 02  LDA $0204       Check for colon ":" as second
91F8  C9 3A     CMP #$3A        character of filename
91FA  D0 0E     BNE $920A       Drive identifier onhand?
91FC  AD 03 02  LDA $0203       YES—Get and save drive
91FF  AA        TAX             number
9200  29 30     AND #$30        Check for number in ASCII-
9202  C9 30     CMP #$30        numbers
9204  D0 04     BNE $920A       Is there a number?
9206  E0 31     CPX #$31        YES—Compare with '1'
```

```
9208   F0 1C      BEQ $9226        Drive 1 switched over?
920A2  AD 03 02   LDA $0203        YES-Compare with '0'
920D   C9 3A      CMP #$3A         Compare with ':'
920F   D0 04      BNE $9215        Is there also a colon?
9211   CE 74 02   DEC $0274        YES-Abbreviate length of filename
9214   C8         INY              Pointer to next buffer byte
92151  A2 00      LDX #$00         Pointer to begin. of input buffer
92171  B9 00 02   LDA $0200,Y      Shift filename to
921A   9D 00 02   STA $0200,X      beginning of buffer
921D   C8         INY              Turn buffer pointer to
921E   E8         INX              next character
921F   EC 74 02   CPX $0274        Compare with end of filename
9222   D0 F3      BNE $9217        Entire name already shifted?
9224   18         CLC              YES-Flag for 'Name error-free'
9225   24         .byte $24        Jump to next byte (bit command)
92261  38         SEC              Flag for bad drive declaration
9227   60         RTS              Return from this subroutine
-----------------------------------------------------------------
[9125/912D/9150/9162/9169/9170/917F/9187/919C/91B2/A9EA]
Byte given over 1571 bus for Fast Load
9228   AD 00 18   LDA $1800        Get bus control register
922B   CD 00 18   CMP $1800        and wait for constant status
922E   D0 F8      BNE $9228        No changes?
9230   29 FF      AND #$FF         YES-Set processor flag (N/Z)
9232   30 17      BMI $924B        Is ATN input set?
9234   45 37      EOR $37          NO-Get bus status flag and check
9236   29 04      AND #$04         with anticipated Clock status
9238   F0 EE      BEQ $9228        Clock changed since last time?
923A   8E 0C 40   STX $400C        YES-Write byte in output register
923D   A5 37      LDA $37          Get bus status and
923F   49 04      EOR #$04         set flag for 'Clock input status'
9241   85 37      STA $37          to next value
9243   A9 08      LDA #$08         Test flag for 'Output register
92451  2C 0D 40   BIT $400D        empty'
9248   F0 FB      BEQ $9245        Is byte transferred?
924A   60         RTS              YES-Return from this subroutine
924B1  4C B3 A7   JMP $A7B3        ATN command working
-----------------------------------------------------------------
[8FB0/BF69]
Compute ROM checksum and test ROM
924E   08         PHP              Retain processor status
924F   78         SEI              Disable bus/controller interrupt
9250   A2 00      LDX #$00         Clear result register
9252   86 00      STX $00          for checksum to
9254   86 01      STX $01          be computed
9256   A9 03      LDA #$03         Set starting address of
9258   85 75      STA $75          ROM (low-byte)
```

```
925A   A8         TAY           Set pointer
925B   A9 80      LDA #$80      Determine high-byte of
925D   85 76      STA $76       ROM address
925F²  B1 75      LDA ($75),Y   Get byte from ROM
9261   85 02      STA $02       and save it
9263   A2 08      LDX #$08      Number of bits per byte
9265¹  A5 02      LDA $02       Get ROM byte and isolate
9267   29 01      AND #$01      a bit
9269   85 03      STA $03       Take bit into temporary storage
926B   A5 01      LDA $01       Add bit 15 of
926D   10 02      BPL $9271     checksum register
926F   E6 03      INC $03       to it
9271¹  6A         ROR A         Add bit 11
9272   90 02      BCC $9276     of checksum register
9274   E6 03      INC $03       to it
9276¹  6A         ROR A         Bit 8 of 16-bit
9277   6A         ROR A         checksum register
9278   6A         ROR A         in $00 and $01;
9279   90 02      BCC $927D     compute for
927B   E6 03      INC $03       temporary storage
927D¹  A5 00      LDA $00       Get bit 6
927F   2A         ROL A         of checksum register
9280   2A         ROL A         and add
9281   90 02      BCC $9285     to temporary
9283   E6 03      INC $03       storage
9285¹  66 03      ROR $03       Move checksum registers one bit
9287   26 00      ROL $00       to the left; transfer bit 0 into
9289   26 01      ROL $01       free area
928B   66 02      ROR $02       Go to next bit of ROM byte
928D   CA         DEX           Number of bits per byte
928E   D0 D5      BNE $9265     Entire byte handled?
9290   E6 75      INC $75       YES—Turn pointer to current
9292   D0 CB      BNE $925F     byte in ROM
9294   E6 76      INC $76       to next position
9296   D0 C7      BNE $925F     Reached end address $FFFF?
9298   88         DEY           YES—Set pointer
9299   88         DEY           back to
929A   88         DEY           zero
929B   A5 00      LDA $00       Test first byte computed against
929D   CD 00 80   CMP $8000     correct checksum
92A0   D0 11      BNE $92B3     Error?
92A2   A5 01      LDA $01       NO—Test 2nd byte computed against
92A4   CD 01 80   CMP $8001     correct checksum
92A7   D0 0A      BNE $92B3     Checksum error?
92A9   84 00      STY $00       NO—Clear checksum register
92AB   84 01      STY $01       and the
92AD   84 02      STY $02       different temporary storage
```

```
92AF   84 03      STY $03         areas
92B1   28         PLP             Re-establish processor status
92B2   60         RTS             Return from this subroutine
92B3²  A2 03      LDX #$03        Initialize flag for
92B5   86 6F      STX $6F         hardware error
92B7   4C 71 EA   JMP $EA71       Show hardware error (LED blinks)
--------------------------------------------------------------
[9E08/9E11/BF09]
1571 jobloop
92BA   BA         TSX             Save current stack
92BB   86 49      STX $49         pointer
92BD   2C 04 1C   BIT $1C04       Timer re-set
92C0   AD 0C 1C   LDA $1C0C       CA2 output 'SOE'
92C3   09 0E      ORA #$0E        (Serial Output Enable)
92C5   8D 0C 1C   STA $1C0C       set to high
92C8   A0 05      LDY #$05        Number of buffers
92CA¹  B9 00 00   LDA $0000,Y     Get jobcode of buffer
92CD   30 06      BMI $92D5       Is jobcode onhand?
92CF   88         DEY             NO—Test next buffer
92D0   10 F8      BPL $92CA       All buffers tested out?
92D2   4C CA 99   JMP $99CA       YES—Execute stepper commands
92D5¹  C9 88      CMP #$88        Jobcode'Read sector on same trak'
92D7   D0 03      BNE $92DC       Identical?
92D9   4C 0D 96   JMP $960D       YES—Read sector in buffer
92DC¹  C9 D0      CMP #$D0        Jobcode for 'Execute program'
92DE   D0 03      BNE $92E3       Identical?
92E0   4C A2 93   JMP $93A2       YES—Start program in buffer
92E3¹  29 01      AND #$01        Get number of desired drive
92E5   F0 07      BEQ $92EE       Drive 0 chosen?
92E7   84 3F      STY $3F         NO—Save buffer number
92E9   A9 0F      LDA #$0F        Display
92EB   4C B5 99   JMP $99B5       'Drive not Ready' error message
92EE¹  AA         TAX             Save drive number and test
92EF   C5 3E      CMP $3E         against active drive
92F1   F0 08      BEQ $92FB       Identical?
92F3   85 3E      STA $3E         NO—Then reset current drive
92F5   20 7E F9   JSR $F97E       Switch drive number on
92F8   4C CA 99   JMP $99CA       Execute stepper command
92FB¹  A5 20      LDA $20         Get drive status
92FD   30 03      BMI $9302       Is drive ready?
92FF   0A         ASL A           YES—Test stepper motor flag
9300   10 03      BPL $9305       Is head still moving?
9302¹  4C CA 99   JMP $99CA       YES—Execute stepper function
9305¹  A9 20      LDA #$20        Set drive status flag for
9307   85 20      STA $20         'Motor on/Drive ready'
9309   A0 05      LDY #$05        Number of buffers
930B   84 3F      STY $3F         Choose current buffer
```

```
930D¹ 20 D1 93   JSR $93D1        Set buffer pointer & get jobcode
9310  30 1A      BMI $932C        Is a job onhand?
9312² C6 3F      DEC $3F          NO—Go to next buffer
9314  10 F7      BPL $930D        All buffers already checked?
9316  A4 41      LDY $41          Get buffer number of last job
9318  20 D3 93   JSR $93D3        Set buffer pointer
931B  A5 42      LDA $42          Save # of track to be controlled
931D  85 4A      STA $4A          as target track
931F  06 4A      ASL $4A          Compute # of half-track steps
9321  A9 60      LDA #$60         Set drive status flag for
9323  85 20      STA $20          'Stepper on/Motor on'
9325  B1 32      LDA ($32),Y      Get and save track of
9327  85 22      STA $22          job
9329  4C CA 99   JMP $99CA        Steer track
932C¹ 29 01      AND #$01         Compare number of chosen drive
932E  C5 3E      CMP $3E          with current drive number
9330  D0 E0      BNE $9312        Identical?
9332  A5 22      LDA $22          Test number of current track
9334  F0 32      BEQ $9368        Is pointer set?
9336  A5 22      LDA $22          YES—Get current track and compare
9338  C9 24      CMP #$24         with maximum tracks +1 (36)
933A  08         PHP              Save result
933B  B1 32      LDA ($32),Y      Compare job track with maximum
933D  C9 24      CMP #$24         tracks + 1
933F  6A         ROR A            Result in bit 7
9340  28         PLP              Previous bit in carry
9341  29 80      AND #$80         Isolate last test result
9343  90 0B      BCC $9350        Is current track on side 2?
9345  30 11      BMI $9358        YES—Is new track on side 1?
9347  A5 22      LDA $22          YES—Compute number of current
9349  E9 23      SBC #$23         track on side 1 and
934B  85 22      STA $22          save it
934D  4C 58 93   JMP $9358        Continue working with track #
9350¹ 10 06      BPL $9358        Is new track on side 2?
9352  A5 22      LDA $22          YES—Calculate current track
9354  69 23      ADC #$23         number on side 2;
9356  85 22      STA $22          save it
9358³ 38         SEC              Figure out difference
9359  B1 32      LDA ($32),Y      between new track
935B  E5 22      SBC $22          and current track
935D  F0 09      BEQ $9368        Head already set to desired trak?
935F  85 42      STA $42          Save # of steps to be moved
9361  A5 3F      LDA $3F          Get number of current buffer
9363  85 41      STA $41          and save it
9365  4C 12 93   JMP $9312        Work on next job
9368² A2 04      LDX #$04         No function [Error--see 7.1.5]
936A  B1 32      LDA ($32),Y      Get number of track
```

```
936C   85 40       STA $40          and save it
936E   C9 24       CMP #$24         Compare with maximum track # +1
9370   A8          TAY              and save result
9371   20 F3 93    JSR $93F3        Go to corresponding side
9374   98          TYA              Repeat track
9375   90 02       BCC $9379        Track on side 2?
9377   E9 23       SBC #$23         YES—Compute absolute track of
9379¹  AA          TAX              that side and save it
937A   BD 08 94    LDA $9408,X      Calculate bitrate of track range
937D   85 43       STA $43          and set it
937F   AD 00 1C    LDA $1C00        Get drive control register
9382   29 9F       AND #$9F         Re-set bits for record rate
9384   05 43       ORA $43          and set into
9386   8D 00 1C    STA $1C00        control register
9389   BD 2B 94    LDA $942B,X      Determine # of sectors per track,
938C   85 43       STA $43          and store
938E   A5 45       LDA $45          Get command bits of jobcode and
9390   C9 40       CMP #$40         test for 'Bump'
9392   F0 1C       BEQ $93B0        Should head be set to track 0?
9394   C9 60       CMP #$60         NO—Check for 'Run program'
9396   F0 0A       BEQ $93A2        Should buffer program be started?
9398   C9 70       CMP #$70         NO—Test for 'Format'
939A   F0 03       BEQ $939F        Should diskette be formatted?
939C   4C 4F 94    JMP $944F        NO—Read sector header
939F¹  4C 29 9B    JMP $9B29        Format diskette
-----------------------------------------------------------------
[92E0/9396]  cf. F36E
Put program in buffer into jobloop
93A2   A5 3F       LDA $3F          Get number of current buffer
93A4   18          CLC              and
93A5   69 03       ADC #$03         calculate physical
93A7   85 31       STA $31          buffer address
93A9   A9 00       LDA #$00         Set low-byte to
93AB   85 30       STA $30          start-of-buffer
93AD   6C 30 00    JMP ($0030)      Run program
-----------------------------------------------------------------
Set head back to track 0 ('Bump')   [cf. F37C]
93B0¹  A9 60       LDA #$60         Set drive status flag for
93B2   85 20       STA $20          'Stepper on/Motor on'
93B4   AD 00 1C    LDA $1C00        Get control register
93B7   29 FC       AND #$FC         and clear stepper control
93B9   8D 00 1C    STA $1C00        bits
93BC   A9 A4       LDA #$A4         Set number of tracks (-36)
93BE   85 4A       STA $4A          the head is capable of moving
93C0   AD B1 01    LDA $01B1        Get flag for current disk side
93C3   30 03       BMI $93C8        Is side 1 chosen?
93C5   A9 01       LDA #$01         YES—Set first track number (1)
```

```
93C7  2C         .byte $2C       Jump to next 2 bytes(bit command)
93C8¹ A9 24      LDA #$24        Save first track of side 2 (36)
93CA  85 22      STA $22         as track number
93CC  A9 01      LDA #$01        Give 'OK'
93CE  4C B5 99   JMP $99B5       return message
------------------------------------------------------------------
[930D/94D3/9527/BF0F/93D3:9318]  cf. F393
Set buffer pointer and get jobcode of buffer
93D1  A4 3F      LDY $3F         Current buffer number
93D3  B9 00 00   LDA $0000,Y     Get matching jobcode and
93D6  48         PHA             save it
93D7  10 14      BPL $93ED       Is there a command onhand?
93D9  29 78      AND #$78        YES—Isolate bits 3-6 and save
93DB  85 45      STA $45         as significant command bits
93DD  98         TYA             Get buffer number and
93DE  0A         ASL A           double it
93DF  69 06      ADC #$06        Set pointer to table of
93E1  85 32      STA $32         track and sector assignments
93E3  A9 00      LDA #$00        to the job
93E5  85 33      STA $33         ($0006-$0011)
93E7  98         TYA             Get buffer number;
93E8  18         CLC             compute physical
93E9  69 03      ADC #$03        memory address
93EB  85 31      STA $31         from that
93ED¹ A0 00      LDY #$00        Put address into
93EF  84 30      STY $30         pointers $30/$31
93F1  68         PLA             Repeat jobcode
93F2  60         RTS             Return from this subroutine
------------------------------------------------------------------
[895C/9371/9B41]
Activate read/write head on current diskette side
93F3  B0 03      BCS $93F8       Is side 2 chosen?
93F5  A9 00      LDA #$00        NO—Control buts for side 1
93F7  2C         .byte $2C       Jump to next 2 bytes
93F8  A9 84      LDA #$84        Control bits/side 2 (%10000100)
93FA  8D B1 01   STA $01B1       Save bits
93FD  AD 0F 18   LDA $180F       Get control register A
9400  29 FB      AND #$FB        and re-set
9402  0D B1 01   ORA $01B1       bits
9405  8D 0F 18   STA $180F       Write value into control register
9408  60         RTS             Return from this subroutine
------------------------------------------------------------------
```

```
[937A]  Control bits for recoding rate of every track

Bit 6 Bit 5 Track range  recording rate
 0     0      31 - 35     31250 Bytes/sec
 0     1      25 - 30     33333 Bytes/sec
 1     0      18 - 24     35714 Bytes/sec
 1     1       1 - 17     38461 Bytes/sec

9409 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
9419 60 40 40 40 40 40 40 40 20 20 20 20 20 20 00 00
9429 00 00 00
-------------------------------------------------------------------
[A82C/A8C2]  Number of sectors per track in Commodore format
942C 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
943C 15 13 13 13 13 13 13 13 12 12 12 12 12 12 11 11
944C 11 11 11
-------------------------------------------------------------------
[939C/97F6]
Look for a sector header
944F  A9 5A     LDA #$5A      Determine number of
9451  85 4B     STA $4B       read attempts (90)
9453¹ 20 54 97  JSR $9754     Wait for next sync-marking
9456¹ 2C 0F 18  BIT $180F     Test 'Byte Ready' signal
9459  30 FB     BMI $9456     Is next byte ready?
945B  AD 01 1C  LDA $1C01     YES—read GCR-byte from diskette
945E  C9 52     CMP #$52      Compare with 'Header' identifier
9460  D0 3E     BNE $94A0     Is it a sector header?
9462  99 24 00  STA $0024,Y   YES—Byte in header buffer
9465  C8        INY           Set buffer pointer to next byte
9466² 2C 0F 18  BIT $180F     Test 'Byte Ready' signal
9469  30 FB     BMI $9466     Is next byte ready?
946B  AD 01 1C  LDA $1C01     YES—Read GCR-byte from diskette
946E  99 24 00  STA $0024,Y   Byte in header buffer
9471  C8        INY           Set buffer pointer to next byte
9472  C0 08     CPY #$08      Number of header bytes
9474  D0 F0     BNE $9466     entire header read?
9476  20 2F 95  JSR $952F     YES—Blockheader / GCR to binary
9479  A0 04     LDY #$04      Number of relevant header bytes
947B  A9 00     LDA #$00      Compute checksum of bytes:
947D¹ 59 16 00  EOR $0016,Y   Compute header byte
9480  88        DEY           Pointer to next byte
9481  10 FA     BPL $947D     All bytes figured out?
9483  C9 00     CMP #$00      Compare with 'Correct' value
9485  D0 30     BNE $94B7     Error-free checksum?
9487  A5 18     LDA $18       YES—Set track number from header
9489  85 22     STA $22       as current track number
948B  A5 45     LDA $45       Get jobcode command bits & check
```

```
948D    C9 30      CMP #$30        code for 'Look for sector'
948F    F0 18      BEQ $94A9       Should sectorheader be sought?
9491    A5 12      LDA $12         NO—Compare ID from sectorheader
9493    C5 16      CMP $16         with current ID
9495    D0 1D      BNE $94B4       Identical?
9497    A5 13      LDA $13         Test next ID
9499    C5 17      CMP $17         character
949B    D0 17      BNE $94B4       Run into a disk exchange?
949D    4C BC 94   JMP $94BC       NO—Get next job
94A0[1] C6 4B      DEC $4B         Number of read attempts
94A2    D0 AF      BNE $9453       Still need to do search?
94A4    A9 02      LDA #$02        NO—'Header Not Found' error #
94A6    20 B5 99   JSR $99B5       Output error message
94A9[1] A5 16      LDA $16         Take current ID
94AB    85 12      STA $12         from header (1st character)
94AD    A5 17      LDA $17         Take current ID
94AF    85 13      STA $13         from header (2nd character)
94B1    A9 01      LDA #$01        Number for 'Ok' message
94B3    2C         .byte $2C       Jump to next 2 bytes(bit command)
94B4[2] A9 0B      LDA #$0B        'ID Mismatch' error number
94B6    2C         .byte $2C       Jump to next 2 bytes(bit command)
94B7[1] A9 09      LDA #$09        'Read Error (27)' error number
94B9    4C B5 99   JMP $99B5       Give return message
-------------------------------------------------------------------
[949D]  cf. F423
Get next job (Sector optimizing)
Optimum is a state of > 6 (read) or 9-12 sectors (write)
94BC    A9 7F      LDA #$7F        Initialize pointer for difference
94BE    85 4C      STA $4C         to next job
94C0    A5 19      LDA $19         Compare sector number
94C2    18         CLC             from blockheader
94C3    69 02      ADC #$02        with maximum
94C5    C5 43      CMP $43         sector number
94C7    90 02      BCC $94CB       Is number in allowable range?
94C9    E5 43      SBC $43         NO—Remove max. number and
94CB[1] 85 4D      STA $4D         save new sector number
94CD    A2 05      LDX #$05        Set buffer 5
94CF    86 3F      STX $3F         as current buffer
94D1    A2 FF      LDX #$FF        Buffer pointer value
94D3[1] 20 D1 93   JSR $93D1       Set buffer pointer & get jobcode
94D6    10 43      BPL $951B       Is a jobcode onhand?
94D8    29 01      AND #$01        YES—Determine corresponding drive
94DA    C5 3E      CMP $3E         and compare with current drive
94DC    D0 3D      BNE $951B       Identical?
94DE    A0 00      LDY #$00        YES—Pntr to params from buffer0
94E0    B1 32      LDA ($32),Y     Get job track for buffer 0
94E2    C5 40      CMP $40         Compare with last track
```

```
94E4   D0 35      BNE $951B       Identical?
94E6   A5 45      LDA $45         YES-Get command bits of jobcode &
94E8   C9 60      CMP #$60        test for 'Execute program' code
94EA   F0 0C      BEQ $94F8       Identical?
94EC   A0 01      LDY #$01        NO-Pntr to params from buffer0
94EE   38         SEC             Get sector number of job for
94EF   B1 32      LDA ($32),Y     for buffer 1
94F1   E5 4D      SBC $4D         Test for optimal sectors computed
94F3   10 03      BPL $94F8       Is new sector number smaller?
94F5   18         CLC             NO-Calculate number of sectors up
94F6   65 43      ADC $43         to this sector and compare
94F8²  C5 4C      CMP $4C         with last difference
94FA   B0 1F      BCS $951B       Is new value smaller or greater?
94FC   48         PHA             YES-Save sector difference
94FD   A5 45      LDA $45         Check command bits of jobcode
94FF   F0 15      BEQ $9516       Should sector be read?
9501   68         PLA             NO-Get difference again and
9502   C9 09      CMP #$09        compare with 9
9504   90 15      BCC $951B       Is value smaller?
9506   C9 0C      CMP #$0C        NO-Compare with 13
9508   B0 11      BCS $951B       Is difference less than 13?
950A¹  85 4C      STA $4C         YES-Save new sector difference
950C   A5 3F      LDA $3F         Get number of current
950E   AA         TAX             buffer and
950F   18         CLC             from it compute
9510   69 03      ADC #$03        the appropriate physical
9512   85 31      STA $31         memory address
9514   D0 05      BNE $951B       Jump to $951B
9516¹  68         PLA             Repeat sector difference &
9517   C9 06      CMP #$06        compare with 6
9519   90 EF      BCC $950A       Is difference less?
951B⁷  C6 3F      DEC $3F         NO-Turn pointer to next buffer
951D   10 B4      BPL $94D3       All buffers tested?
951F   8A         TXA             YES-Buffer number of next job
9520   10 03      BPL $9525       Optimal job found?
9522   4C CA 99   JMP $99CA       execute stepper commands
9525¹  86 3F      STX $3F         Save number of current buffer
9527   20 D1 93   JSR $93D1       Set buffer pointer & get jobcode
952A   A5 45      LDA $45         Determine command bits of jobcode
952C   4C 06 96   JMP $9606       Execute read/write jobs
-------------------------------------------------------------
```

```
[9476]  cf. F497
Convert sector header from GCR to binary
952F  A5 30      LDA $30       Retain low-byte of current buffer
9531  48         PHA           pointer
9532  A5 31      LDA $31       Retain hi-byte of current buffer
9534  48         PHA           pointer
9535  A9 24      LDA #$24      Set buffer pointers $30/$31
9537  85 30      STA $30       to start of buffer
9539  A9 00      LDA #$00      for the current
953B  85 31      STA $31       sector header s
953D  A9 00      LDA #$00      Reset buffer
953F  85 34      STA $34       pointer
9541  20 D9 98   JSR $98D9     Convrt 5 GCRbytes >4 binary bytes
9544  A5 55      LDA $55       Get first converted byte and save
9546  85 18      STA $18       as track number of header
9548  A5 54      LDA $54       Get second converted byte & save
954A  85 19      STA $19       as sector number
954C  A5 53      LDA $53       Get third converted byte and set
954E  85 1A      STA $1A       as header checksum
9550  20 D9 98   JSR $98D9     Convrt 5 GCRbytes >4 binary bytes
9553  A5 52      LDA $52       Set first converted byte
9555  85 17      STA $17       as second ID character
9557  A5 53      LDA $53       Set second converted byte as
9559  85 16      STA $16       first ID character
955B  68         PLA           Get original values of
955C  85 31      STA $31       buffer pointers $30/$31
955E  68         PLA           and
955F  85 30      STA $30       set
9561  60         RTS           Return from this subroutine

--------------------------------------------------------------
9562  FF ...                   Unused
95FF  ... FF                   ROM area
--------------------------------------------------------------
```

```
[960D/98A6]  cf. F50A
Look for data sector and set head to start-of-data
9600  20 0F 97   JSR $970F       Look for sector header
9603  4C 54 97   JMP $9754       Wait for next Sync-marking
-------------------------------------------------------------------
[952C]  cf. F4CA
Read Commodore sectors when jobcode = $80 (command bits $00)
9606  C9 00      CMP #$00        Test for 'Read sector' jobcode
9608  F0 03      BEQ $960D       Identical?
960A  4C 6E 97   JMP $976E       NO—Continue jobcode test
-------------------------------------------------------------------
[92D9/9608]
Read sector
960D  20 00 96   JSR $9600       Look for data block
9610¹ 2C 0F 18   BIT $180F       Wait for 'Byte ready'
9613  30 FB      BMI $9610       signal
9615  AD 01 1C   LDA $1C01       Read byte from diskette
9618  AA         TAX             and save it
9619  BD 0D A0   LDA $A00D,X     Get binary equivalent and
961C  85 52      STA $52         save it
961E  8A         TXA             Repeat original byte and
961F  29 07      AND #$07        expand first GCR part
9621  85 53      STA $53         Save byte
9623¹ 2C 0F 18   BIT $180F       wait for 'Byte ready'
9626  30 FB      BMI $9623       signal
9628  AD 01 1C   LDA $1C01       Read byte from diskette
962B  85 54      STA $54         and save it
962D  29 C0      AND #$C0        Get last 2 bits of 2nd GCR-byte
962F  05 53      ORA $53         and add in first 3 bits
9631  AA         TAX             (1st part:bits0-2;2ndprt:bits6-7)
9632  BD 0D 9F   LDA $9F0D,X     Get binary equivalent, OR with
9635  05 52      ORA $52         previous half-byte of 1st byte
9637  48         PHA             Save byte as data blok identifier
9638  4C 67 96   JMP $9667       Read data part
-------------------------------------------------------------------
[963E/96D4]
Read GCR-bytes from diskette and put into buffer as binary bytes
963B  2C 0F 18   BIT $180F       Wait for 'Byte ready'
963E  30 FB      BMI $963B       signal
9640  AD 01 1C   LDA $1C01       Read byte from diskette and
9643  AA         TAX             save it
9644  BD 0D A0   LDA $A00D,X     Determine binary equivalent and
9647  85 52      STA $52         store away temporarily
9649  8A         TXA             Repeat original data byte spread
964A  29 07      AND #$07        out first GCR-byte
964C  85 53      STA $53         Save part of 2nd GCR-byte
```

```
964E[1] 2C 0F 18   BIT $180F      Wait for 'Byte ready'
9651    30 FB      BMI $964E      signal
9653    AD 01 1C   LDA $1C01      Read byte from diskette
9656    85 54      STA $54        and save it
9658    29 C0      AND #$C0       Get last part of 2nd GCR byte
965A    05 53      ORA $53        and combine with 1st part
965C    AA         TAX            (1stpart:bits 0-2;2ndprt:bits6-7)
965D    BD 0D 9F   LDA $9F0D,X    Get corresponding half-byte &
9660    05 52      ORA $52        combine previous half-byte
9662    91 30      STA ($30),Y    Write binary byte to buffer
9664    C8         INY            turn buffer pointer to next byte
9665    F0 70      BEQ $96D7      Reached end-of-buffer?
9667[1] A5 54      LDA $54        NO—Get next GCR-byte & determine
9669    AA         TAX            upper half-byte of
966A    BD 0D A1   LDA $A10D,X    equivalent binary byte;
966D    85 52      STA $52        save it
966F    8A         TXA            Repeat original GCR-byte and form
9670    29 01      AND #$01       first part of next GCR-bytes;
9672    85 54      STA $54        save it
9674[1] 2C 0F 18   BIT $180F      Wait for 'Byte ready'
9677    30 FB      BMI $9674      signal
9679    AD 01 1C   LDA $1C01      Read byte from diskette
967C    85 55      STA $55        and save it
967E    29 F0      AND #$F0       Determine 2nd part of GCR-byte
9680    05 54      ORA $54        and connect with 1st part
9682    AA         TAX            (1st part:bit0; 2nd part:bits4-7)
9683    BD 0F 9F   LDA $9F0F,X    Get corresponding half-byte and
9686    05 52      ORA $52        form next binary byte
9688    91 30      STA ($30),Y    Write byte into buffer
968A    C8         INY            Turn buffer pointer to next byte
968B    A5 55      LDA $55        Setup first part of
968D    29 0F      AND #$0F       next GCR-byte and
968F    85 55      STA $55        save it
9691[1] 2C 0F 18   BIT $180F      Wait for 'Byte ready'
9694    30 FB      BMI $9691      signal
9696    AD 01 1C   LDA $1C01      Read byte from diskette
9699    85 3A      STA $3A        and save it
969B    29 80      AND #$80       Set second part of GCR-byte and
969D    05 55      ORA $55        combine with first part
969F    AA         TAX            (1st part:bits 0-3;2nd part:bit7)
96A0    BD 1D 9F   LDA $9F1D,X    Determine & temporarily store 1st
96A3    85 52      STA $52        half-byte of next binary value
96A5    A5 3A      LDA $3A        Repeat original GCR-value and
96A7    AA         TAX            get second half-byte
96A8    BD 0D A2   LDA $A20D,X    of equivalent binary byte
```

```
96AB  05 52     ORA $52       Add in first part and write byte
96AD  91 30     STA ($30),Y   to buffer
96AF  C8        INY           Turn buffer pointer to next byte
96B0  8A        TXA           Set up first part
96B1  29 03     AND #$03      of next GCR-byte
96B3  85 3A     STA $3A       and save it
96B5¹ 2C 0F 18  BIT $180F     Wait for 'Byte ready'
96B8  30 FB     BMI $96B5     signal
96BA  AD 01 1C  LDA $1C01     read byte from diskette and
96BD  85 53     STA $53       store away temporarily
96BF  29 E0     AND #$E0      Isolat 2nd part of GCR-byte,
96C1  05 3A     ORA $3A       and combine with 1st part
96C3  AA        TAX           (1stpart:bits 0-1;2ndprt:bits5-7)
96C4  BD 2A 9F  LDA $9F2A,X   Determine and save first binary
96C7  85 52     STA $52       half-byte
96C9  A5 53     LDA $53       Repeat originalGCR-value and
96CB  AA        TAX           get second part of
96CC  BD 0D A3  LDA $A30D,X   binary byte
96CF  05 52     ORA $52       Include first half-byte
96D1  91 30     STA ($30),Y   Write byte to buffer
96D3  C8        INY           Set buffer pointer >next position
96D4  4C 3B 96  JMP $963B     Next 5 GCRbytes in 4 binary bytes
----------------------------------------------------------------
[9665]
End of buffer reached
96D7  A5 54     LDA $54       Get last GCR-byte and determine
96D9  AA        TAX           first half-byte of
96DA  BD 0D A1  LDA $A10D,X   next binary byte
96DD  85 52     STA $52       save it
96DF  8A        TXA           Repeat original GCR-value and
96E0  29 01     AND #$01      isolate first part of next GCR-
96E2  85 54     STA $54       byte
96E4¹ 2C 0F 18  BIT $180F     Wait for 'Byte ready'
96E7  30 FB     BMI $96E4     signal
96E9  AD 01 1C  LDA $1C01     Read byte from diskette
96EC  29 F0     AND #$F0      and get second part of GCR-byte
96EE  05 54     ORA $54       Combine with first part
96F0  AA        TAX           (1st part:bit 0;2nd part:bits4-7)
96F1  BD 0F 9F  LDA $9F0F,X   Determine 2nd part of binary byte
96F4  05 52     ORA $52       and form final binary byte
96F6  85 53     STA $53       Save value as checksum
96F8  68        PLA           Repeat data block identifier and
96F9  C5 47     CMP $47       test it
96FB  D0 0A     BNE $9707     Is identifier correct?
96FD  20 E9 F5  JSR $F5E9     YES—Compute buffer checksum
9700  C5 53     CMP $53       Test against checksum to be given
```

```
9702  F0 06     BEQ $970A     Is a read error taking place?
9704  A9 05     LDA #$05      YES-Error # for 'Read Error (23)'
9706  2C        .byte $2C     Jump to next 2 bytes(bit command)
9707¹ A9 04     LDA #$04      Error # for 'Read Error (22)'
9709  2C        .byte $2C     Jump to next 2 bytes(bit command)
970A¹ A9 01     LDA #$01      Value for 'Ok' message
970C  4C B5 99  JMP $99B5     Give return message
-------------------------------------------------------------------
[9600/9789/98CE]  cf. F510
Look for sector header
970F  A5 12     LDA $12       Write current ID (1st character)
9711  85 16     STA $16       in buffer for Sector header
9713  A5 13     LDA $13       Write current ID (2nd character)
9715  85 17     STA $17       in buffer for sector header
9717  A0 00     LDY #$00      Re-set buffer pointer
9719  B1 32     LDA ($32),Y   Get track of current job and take
971B  85 18     STA $18       in Header buffer
971D  C8        INY           Buffer pointer to next byte
971E  B1 32     LDA ($32),Y   take number of current sector
9720  85 19     STA $19       in header
9722  A9 00     LDA #$00      Compute checksum:
9724  45 16     EOR $16       ID 1
9726  45 17     EOR $17       ID 2
9728  45 18     EOR $18       Track number
972A  45 19     EOR $19       Sector number
972C  85 1A     STA $1A       Checksum in header buffer
972E  20 34 F9  JSR $F934     Convert header into GCR-values
9731  A9 5A     LDA #$5A      Set number of read
9733  85 4B     STA $4B       attempts (90)
9735¹ 20 54 97  JSR $9754     Wait for next Sync-marking
9738¹ B9 24 00  LDA $0024,Y   Get byte from header buffer
973B¹ 2C 0F 18  BIT $180F     Wait for 'Byte ready'
973E  30 FB     BMI $973B     signal
9740  CD 01 1C  CMP $1C01     Compare with byte on diskette
9743  D0 06     BNE $974B     Identical?
9745  C8        INY           YES-Compare next byte
9746  C0 08     CPY #$08      Number of bytes to a header
9748  D0 EE     BNE $9738     Entire header compared?
974A  60        RTS           YES-Return from this subroutine
974B¹ C6 4B     DEC $4B       Try again
974D  D0 E6     BNE $9735     Number of read attempts ended?
974F  A9 02     LDA #$02      YES-Error # for 'Read Error (21)'
9751  4C B5 99  JMP $99B5     Give return message
-------------------------------------------------------------------
```

```
[9453/9603/9735/9CDD/9D08/BF21]  cf. F556
Wait for next Sync-marking
9754  A2 0F      LDX #$0F        Set attempt counter
9756  A0 00      LDY #$00        (ca. 47 / 23 ms)
9758  2C 00 1C   BIT $1C00       Test 'Sync' signal
975B  10 0B      BPL $9768       Is Sync set?
975D  88         DEY             NO—Decrement counter
975E  D0 F8      BNE $9758       Counter already running
9760  CA         DEX             Decrement counter
9761  D0 F5      BNE $9758       256 cycles passed?
9763  A9 03      LDA #$03        Error number for 'sync not found'
9765  4C B5 99   JMP $99B5       Give return message
9768  AD 01 1C   LDA $1C01       Initialize 'Byte ready' (CA1)
976B  A0 00      LDY #$00        Clear register
976D  60         RTS             Return from this subroutine
----------------------------------------------------------------
[960A]  cf. F56E
Write sector, when jobcode =$90 (command bit $10)
976E  C9 10      CMP #$10        Test for 'Write' jobcode
9770  F0 03      BEQ $9775       Should sector be written?
9772  4C 98 98   JMP $9898       NO—Continue jobcode test
----------------------------------------------------------------
Write sector
9775[1] 20 E9 F5 JSR $F5E9       Compute buffer checksum and
9778  85 3A      STA $3A         save it
977A  AD 00 1C   LDA $1C00       Get drive control register and
977D  29 10      AND #$10        test 'Write Protect' bit
977F  D0 05      BNE $9786       Is write-protect set?
9781  A9 08      LDA #$08        YES—'Write protect on' error #
9783  4C B5 99   JMP $99B5       Set return message
9786[1] 20 8F F7 JSR $F78F       Convert buffer contents to GCR
9789  20 0F 97   JSR $970F       Look for blockheader
978C  A0 09      LDY #$09        Gap bytes until data block
978E[2] 2C 0F 18 BIT $180F       wait for 'Byte ready'
9791  30 FB      BMI $978E       signal
9793  2C 00 1C   BIT $1C00       Get head ready again
9796  88         DEY             Another byte
9797  D0 F5      BNE $978E       Gap already jumped over?
9799  A9 FF      LDA #$FF        YES—Set head register for
979B  8D 03 1C   STA $1C03       output
979E  AD 0C 1C   LDA $1C0C       Get control register and
97A1  29 1F      AND #$1F        place head circuitry
97A3  09 C0      ORA #$C0        in write mode
97A5  8D 0C 1C   STA $1C0C       (CB2 to low)
97A8  A9 FF      LDA #$FF        Sync-marking value
97AA  A0 05      LDY #$05        Number of Sync-bytes
```

```
97AC    8D 01 1C   STA $1C01       Write byte to diskette
97AF[2] 2C 0F 18   BIT $180F       Wait for 'Byte Ready'
97B2    30 FB      BMI $97AF       signal
97B4    2C 00 1C   BIT $1C00       Re-set 'Byte Ready'
97B7    88         DEY             Next Sync-byte
97B8    D0 F5      BNE $97AF       Entire marking already written?
97BA    A0 BB      LDY #$BB        YES—Buffer pointr to status bufer
97BC[1] B9 00 01   LDA $0100,Y     Get byte from buffer
97BF[1] 2C 0F 18   BIT $180F       wait for 'Byte ready'
97C2    30 FB      BMI $97BF       signal
97C4    8D 01 1C   STA $1C01       Write GCR-byte to diskette
97C7    C8         INY             Turn pointer to next byte
97C8    D0 F2      BNE $97BC       Entire status buffer on diskette?
97CA[1] B1 30      LDA ($30),Y     YES—Get byte from current buffer
97CC[1] 2C 0F 18   BIT $180F       wait for 'Byte ready'
97CF    30 FB      BMI $97CC       signal
97D1    8D 01 1C   STA $1C01       Write GCR-byte to diskette
97D4    C8         INY             Set buffer pointer to next byte
97D5    D0 F3      BNE $97CA       Entire buffer written?
97D7[1] 2C 0F 18   BIT $180F       Wait for 'Byte ready'
97DA    30 FB      BMI $97D7       signal, until byte is written
97DC    AD 0C 1C   LDA $1C0C       Switch head electronics
97DF    09 E0      ORA #$E0        to read mode
97E1    8D 0C 1C   STA $1C0C       (CB2 to high)
97E4    A9 00      LDA #$00        Set head register
97E6    8D 03 1C   STA $1C03       to input
97E9    20 F9 97   JSR $97F9       Convert buffer from GCR to binary
97EC    A4 3F      LDY $3F         Number of current buffer
97EE    B9 00 00   LDA $0000,Y     Get jobcode and convert
97F1    49 30      EOR #$30        into 'Test sector'
97F3    99 00 00   STA $0000,Y     Set new jobcode
97F6    4C 4F 94   JMP $944F       Look for sector header
--------------------------------------------------------------------
[97E9/99BE]  cf. F5F2
Convert current buffer &status buffer($01BB-$1FF)from GCR to binary
97F9    A9 00      LDA #$00        Initialize lo-byte of pointer for
97FB    85 2E      STA $2E         current data buffer and
97FD    85 30      STA $30         status buffer
97FF    85 4F      STA $4F         Hold momentary value of pointer
9801    A5 31      LDA $31         for current data buffer in
9803    85 4E      STA $4E         in $4E/$4F
9805    A9 01      LDA #$01        Turn buffer pointer to
9807    85 31      STA $31         status buffer
9809    85 2F      STA $2F         (high-byte)
980B    A9 BB      LDA #$BB        Turn buffer pointr/conver routine
980D    85 34      STA $34         to start of status buffer ($1BB)
```

```
980F   85 36      STA $36        Set pointer to current byte pos
9811   20 D9 98   JSR $98D9      Convert 5 GCR to 4 binary bytes
9814   A5 52      LDA $52        Get identifier of data block
9816   85 38      STA $38        and save it
9818   A4 36      LDY $36        Get buffer pointer-next bin byte
981A   A5 53      LDA $53        Get data byte and write
981C   91 2E      STA ($2E),Y    to buffer
981E   C8         INY            Set buffer pointer to next byte
981F   A5 54      LDA $54        Get data byte and
9821   91 2E      STA ($2E),Y    write to buffer
9823   C8         INY            Set buffer pointer to next byte
9824   A5 55      LDA $55        Get data byte and
9826   91 2E      STA ($2E),Y    write to buffer
9828   C8         INY            Set buffer pointer to next byte
9829   84 36      STY $36        Save buffer pointer
982B1  20 D9 98   JSR $98D9      Convert 5 GCR-bytes-4 bin bytes
982E   A4 36      LDY $36        Repeat buffer pointer
9830   A5 52      LDA $52        Get data byte and write
9832   91 2E      STA ($2E),Y    to buffer
9834   C8         INY            Set buffer pointer to next byte
9835   A5 53      LDA $53        Get databyte and
9837   91 2E      STA ($2E),Y    write to buffer
983A   F0 0E      BEQ $984A      End of status buffer reached?
983C   A5 54      LDA $54        NO—Get databyte and
983E   91 2E      STA ($2E),Y    write to buffer
9840   C8         INY            set buffer pointer to next byte
9841   A5 55      LDA $55        Get databyte and
9843   91 2E      STA ($2E),Y    write to buffer
9845   C8         INY            Set buffer pointer to next byte
9846   84 36      STY $36        Save buffer pointer
9848   D0 E1      BNE $982B      Reached end of status buffer?
984A1  A5 54      LDA $54        YES—Get converted binary byte and
984C   91 30      STA ($30),Y    write to data buffer
984E   C8         INY            Set buffer pointer to next byte
984F   A5 55      LDA $55        Get converted binary byte and
9851   91 30      STA ($30),Y    write to data buffer
9853   C8         INY            Set buffer pointer to next byte
9854   84 36      STY $36        Save buffer pointer
98561  20 D9 98   JSR $98D9      Conver 5 GCR-Bytes-4 bin bytes
9859   A4 36      LDY $36        Get buffer pointer again
985B   A5 52      LDA $52        Get converted binary byte and
985D   91 30      STA ($30),Y    write to data buffer
985F   C8         INY            Set buffer pointer to next byte
9860   A5 53      LDA $53        Get converted binary byte and
9862   91 30      STA ($30),Y    write to data buffer
9864   C8         INY            Set buffer pointer to next byte
```

```
9865  A5 54     LDA $54        Get converted binary byte and
9867  91 30     STA ($30),Y    write to data buffer
9869  C8        INY            Set buffer pointer to next byte
986A  A5 55     LDA $55        Get converted binary byte and
986C  91 30     STA ($30),Y    write to data buffer
986E  C8        INY            Set buffer pointer to next byte
986F  84 36     STY $36        Save buffer pointer and
9871  C0 BB     CPY #$BB       Compare with end value
9873  90 E1     BCC $9856      End of data buffer reached?
9875  A9 45     LDA #$45       YES—Set pointer to
9877  85 2E     STA $2E        target address of
9879  A5 31     LDA $31        the following shift
987B  85 2F     STA $2F        action
987D  A0 BA     LDY #$BA       Shift bytes in data buffer from
987F1 B1 30     LDA ($30),Y    $01-$BB to position $46-$FF in
9881  91 2E     STA ($2E),Y    buffer above
9883  88        DEY            buffer pointer to next byte
9884  D0 F9     BNE $987F      All characters already shifted?
9886  B1 30     LDA ($30),Y    YES—Copy byte $00 to
9888  91 2E     STA ($2E),Y    position $45
988A  A2 BB     LDX #$BB       Pointer to start of status buffer
988C1 BD 00 01  LDA $0100,X    Get byte from status buffer
988F  91 30     STA ($30),Y    and copy into data buffer
9891  C8        INY            turn buffer pointer for Data- and
9892  E8        INX            status buffer to next byte
9893  D0 F7     BNE $988C      Entire buffer copied?
9895  86 50     STX $50        YES—Clr flag fr'buffer in GCR'(0)
9897  60        RTS            Return from this subroutine
------------------------------------------------------------
```

```
[9772]  cf. F691
Sector verify, when jobcode = $A0 (Command bit $20)
9898  C9 20      CMP #$20       Test for jobcode:'Sector verify'
989A  F0 02      BEQ $989E      Should sector be verified?
989C  D0 30      BNE $98CE      N—Jump to $98CE
-------------------------------------------------------------------
Sector verify
989E1 20 E9 F5   JSR $F5E9      Compute buffer checksum
98A1  85 3A      STA $3A        and save it
98A3  20 8F F7   JSR $F78F      Convert buffer from binary to GCR
98A6  20 00 96   JSR $9600      Look for sector header
98A9  A0 BB      LDY #$BB       Turn pointer to status buffer
98AB1 B9 00 01   LDA $0100,Y    Get byte from buffer
98AE1 2C 0F 18   BIT $180F      Wait for 'byte ready'
98B1  30 FB      BMI $98AE      signal and compare
98B3  4D 01 1C   EOR $1C01      with byte from diskette
98B6  D0 1C      BNE $98D4      Identical?
98B8  C8         INY            YES—Buffer pointer to next byte
98B9  D0 F0      BNE $98AB      Entire status buffer compared?
98BB1 B1 30      LDA ($30),Y    byte from data buffer
98BD1 2C 0F 18   BIT $180F      wait for 'byte ready'
98C0  30 FB      BMI $98BD      signal
98C2  4D 01 1C   EOR $1C01      Compare with byte from diskette
98C5  D0 0D      BNE $98D4      Identical?
98C7  C8         INY            YES—Buffer pointer to next byte
98C8  C0 FD      CPY #$FD       Compare pointer with end value
98CA  D0 EF      BNE $98BB      Reached end of data buffer?
98CC  F0 03      BEQ $98D1      YES—Jump to $98D1
98CE1 20 0F 97   JSR $970F      Look for next sector header
98D11 A9 01      LDA #$01       Number for 'Ok' message
98D3  2C         .byte $2C      Jump to next 2 bytes
98D42 A9 07      LDA #$07       Set 'Verify Error' error number
98D6  4C B5 99   JMP $99B5      Send return message
-------------------------------------------------------------------
[9541/9550/9811/982B/9856/9979/9993/BF30]  cf. F7E6
Convert 5 GCR-bytes into 4 binary bytes
98D9  A4 34      LDY $34        Get pointer to next GCR-byte
98DB  B1 30      LDA ($30),Y    Get GCR-byte from buffer and
98DD  85 56      STA $56        save as first GCR value
98DF  29 07      AND #$07       Save 1st part of 2nd
98E1  85 57      STA $57        GCR value
98E3  C8         INY            Pointer to next GCR-byte
98E4  D0 06      BNE $98EC      Reached end of status buffer?
98E6  A5 4E      LDA $4E        YES—set pointer to beginning of
98E8  85 31      STA $31        current data buffer
98EA  A4 4F      LDY $4F        Set pointer to position in buffer
```

```
98EC1 B1 30     LDA ($30),Y   Get GCR-byte from buffer
98EE  85 58     STA $58       and save it
98F0  29 C0     AND #$C0      Get 2nd part of 2nd GCR value;
98F2  05 57     ORA $57       combine with first part
98F4  85 57     STA $57       Save second GCR value
98F6  A5 58     LDA $58       Get original GCR-byte again and
98F8  29 01     AND #$01      get 1st part of 3rd GCR value
98FA  85 59     STA $59       Save value
98FC  C8        INY           Set buffer pointer to next byte
98FD  B1 30     LDA ($30),Y   Get byte from buffer
98FF  AA        TAX           and save it
9900  29 F0     AND #$F0      Find 2nd part of 3rd GCR value;
9902  05 59     ORA $59       combine with first part
9904  85 59     STA $59       Save entire byte
9906  8A        TXA           Get original GCR-byte again, and
9907  29 0F     AND #$0F      save 1st part of
9909  85 5A     STA $5A       4th GCR value
990B  C8        INY           Turn buffer pointer to next byte
990C  B1 30     LDA ($30),Y   Get byte from buffer and
990E  85 5B     STA $5B       save it
9910  29 80     AND #$80      get 2nd part of 4th GCR value and
9912  05 5A     ORA $5A       combine with previous first part
9914  85 5A     STA $5A       Save entire value
9916  A5 5B     LDA $5B       Get original GCR-byte again and
9918  29 03     AND #$03      isolate 1st part of 5th GCR
991A  85 5C     STA $5C       value
991C  C8        INY           Set pointer to next byte
991D  D0 08     BNE $9927     Reached end of status buffer?
991F  A5 4E     LDA $4E       YES—Turn pointer to current
9921  85 31     STA $31       data buffer
9923  A4 4F     LDY $4F       Set pointer in position in
9925  84 30     STY $30       buffer
99271 B1 30     LDA ($30),Y   Get byte from buffer and
9929  85 5D     STA $5D       save it
992B  29 E0     AND #$E0      Get 2nd part of 5th GCR value and
992D  05 5C     ORA $5C       combine with first part
992F  85 5C     STA $5C       Save entire GCR value
9931  C8        INY           Buffer pointer to next character
9932  84 34     STY $34       and save it
9934  A6 56     LDX $56       Get 1st GCR value-find equivalent
9936  BD 0D A0  LDA $A00D,X   most significant binary half-byte
9939  A6 57     LDX $57       Get second GCR value and form
993B  1D 0D 9F  ORA $9F0D,X   least signifcant binary half-byte
993E  85 52     STA $52       Save first converted binary byte
9940  A6 58     LDX $58       Get 3rd GCR value,find equivalent
9942  BD 0D A1  LDA $A10D,X   most significant binary half-byte
```

```
9945  A6 59     LDX $59        Get fourth GCR alue and form
9947  1D 0F 9F  ORA $9F0F,X    least significant bin. half-byte
994A  85 53     STA $53        Save second converted binary byte
994C  A6 5A     LDX $5A        Get 5th GCR value,find equivalent
994E  BD 1D 9F  LDA $9F1D,X    most significant binary half-byte
9951  A6 5B     LDX $5B        Get sixth GCR value and form
9953  1D 0D A2  ORA $A20D,X    least significant half-byte
9956  85 54     STA $54        Save third converted binary byte
9958  A6 5C     LDX $5C        Get 7th GCR value,find equivalent
995A  BD 2A 9F  LDA $9F2A,X    most significant binary half-byte
995D  A6 5D     LDX $5D        Get eighth GCR value and form
995F  1D 0D A3  ORA $A30D,X    least significant bin. half-byte
9962  85 55     STA $55        Save last converted binary byte
9964  60        RTS            Return from this subroutine
------------------------------------------------------------------
[BF27/Routine not used in DOS]  cf. F8E0
Convert status buffer from GR to binary
9965  A9 00     LDA #$00       Pointer to current GCR-byte
9967  85 34     STA $34        set back
9969  85 2E     STA $2E        Clear pointer to target buffer
996B  85 36     STA $36        Pointer to current data position
996D  A9 01     LDA #$01       Turn temp. storage for address
996F  85 4E     STA $4E        of current data buffer
9971  A9 BA     LDA #$BA       to beginning of
9973  85 4F     STA $4F        status buffer
9975  A5 31     LDA $31        Set buffer pointer to value of
9977  85 2F     STA $2F        current data buffer
9979  20 D9 98  JSR $98D9      Convert 5 GCRbytes to 4 bin.bytes
997C  A5 52     LDA $52        Get first binary byte & take as
997E  85 38     STA $38        identifier of data blocks
9980  A4 36     LDY $36        Pointer to current byte
9982  A5 53     LDA $53        Get second converted byte and
9984  91 2E     STA ($2E),Y    write in buffer
9986  C8        INY            Buffer pointer to next byte
9987  A5 54     LDA $54        Get third converted byte and
9989  91 2E     STA ($2E),Y    write to buffer
998B  C8        INY            Buffer pointer to next byte
998C  A5 55     LDA $55        Get last converted byte and
998E  91 2E     STA ($2E),Y    write to buffer
9990  C8        INY            Buffer pointer to next byte
9991[1] 84 36   STY $36        and save it
9993  20 D9 98  JSR $98D9      Convert 5 GCRbytes to 4 bin.bytes
9996  A4 36     LDY $36        Repeat buffer pointer
9998  A5 52     LDA $52        Get 1st converted byte and write
999A  91 2E     STA ($2E),Y    to buffer
999C  C8        INY            Buffer pointer to next byte
```

```
999D  F0 11      BEQ $99B0       Reached end of buffer?
999F  A5 53      LDA $53         N—Get 2nd converted binary byte
99A1  91 2E      STA ($2E),Y     and write to buffer
99A3  C8         INY             Set buffer pointer to next byte
99A4  A5 54      LDA $54         Get third converted byte and
99A6  91 2E      STA ($2E),Y     write to buffer
99A8  C8         INY             Buffer pointer to next byte
99A9  A5 55      LDA $55         Get 3rd converted byte and write
99AB  91 2E      STA ($2E),Y     to buffer
99AD  C8         INY             Buffer pointer to next byte
99AE  D0 E1      BNE $9991       Reached end of buffer?
99B0¹ A5 2F      LDA $2F         YES—Reestablish pointer to
99B2  85 31      STA $31         current data buffer
99B4  60         RTS             Return from this subroutine
-----------------------------------------------------------------
[92EB/93CE/94A6/94B9/970C/9751/9765/9783/9806/904E/9D60/BF15]  cf.
F969
Give return message over job loop
99B5  A4 3F      LDY $3F         Number of current buffer
99B7  99 00 00   STA $0000,Y     Write return message to job reg.
99BA  A5 50      LDA $50         Flag for 'buffer in GCR-Code'
99BC  F0 03      BEQ $99C1       Is the buffer still in GCR ?
99BE  20 F9 97   JSR $97F9       YES—Convert buffer, GCR to binary
99C1¹ 20 8F F9   JSR $F98F       Drive motor off
99C4  A6 49      LDX $49         Redirect
99C6  9A         TXS             stack pointer
99C7  4C C8 92   JMP $92C8       1571 job loop
-----------------------------------------------------------------
[92D2/92F8/9302/9329/9522/9B55/9B64/9D41/9D56/BF72]
Part of jobloop for motor- and stepper control
99CA  AD 07 1C   LDA $1C07       Timer 1 (high-byte)
99CD  8D 05 1C   STA $1C05       re-set
99D0  AD 00 1C   LDA $1C00       Get drive control register
99D3  29 10      AND #$10        and test for 'Write Protect'
99D5  C5 1E      CMP $1E         Compare with last test
99D7  85 1E      STA $1E         and save current status
99D9  D0 07      BNE $99E2       Has 'Write Protect' been changed?
99DB  AD AB 02   LDA $02AB       N—Motor runtime counter
99DE  D0 10      BNE $99F0       Is motor on?
99E0  F0 1C      BEQ $99FE       N—Jump to $99FE
99E2¹ A9 FF      LDA #$FF        Set counter for motor runtime in
99E4  8D AB 02   STA $02AB       disk exchange
99E7  20 64 87   JSR $8764       Motor off
99EA  A9 01      LDA #$01        Set 'Newly initialize diskette'
99EC  85 1C      STA $1C         flag
99EE  D0 0E      BNE $99FE       Jump to $99FE
```

```
99F0¹  CE AB 02   DEC $02AB     Decrement counter f/motor runtime
99F3   D0 09      BNE $99FE     Motor now off?
99F5   A5 20      LDA $20       YES—Get drive status
99F7   C9 00      CMP #$00      Compare with 'motor out'
99F9   D0 03      BNE $99FE     Identical?
99FB   20 70 87   JSR $8770     YES—Drive motor off
99FE⁴  AD FE 02   LDA $02FE     Read error control byte for head
9A01   F0 15      BEQ $9A18     Should head be set to next track?
9A03   C9 02      CMP #$02      N—Test with 'control byte taken'
9A05   D0 07      BNE $9A0E     Is head evenly set?
9A07   A9 00      LDA #$00      Clear control byte
9A09   8D FE 02   STA $02FE     register
9A0C   F0 0A      BEQ $9A18     Jump to $9A18
9A0E¹  85 4A      STA $4A       Set # of steps to be performed
9A10   A9 02      LDA #$02      Set 'Control byte taken'
9A12   8D FE 02   STA $02FE     flag
9A15   4C 56 9A   JMP $9A56     Re-position head
9A18²  A6 3E      LDX $3E       Flag for 'drive aktiv'
9A1A   30 07      BMI $9A23     Is flag set?
9A1C   A5 20      LDA $20       N—Get drive status
9A1E   A8         TAY           and save it
9A1F   C9 20      CMP #$20      Compare with 'Motor on' flag
9A21   D0 03      BNE $9A26     Is drive ready?
9A23²  4C C9 9A   JMP $9AC9     YES—Return from this subroutine
9A26¹  C6 48      DEC $48       Motor delay counter
9A28   D0 1C      BNE $9A46     Is motor out of turn mode?
9A2A   98         TYA           YES—Get drive status
9A2B   10 04      BPL $9A31     Flag for 'Motor not ready' set?
9A2D   29 7F      AND #$7F      YES—Clear
9A2F   85 20      STA $20       flag
9A31¹  29 10      AND #$10      Flag for 'Motor in off phase'
9A33   F0 11      BEQ $9A46     Should motor be turned off?
9A35   C6 35      DEC $35       Jobloop calls yet to be performed
9A37   D0 0D      BNE $9A46     Jobloop called again?
9A39   20 70 87   JSR $8770     N—Drive motor off
9A3C   A9 FF      LDA #$FF      Clear 'drive active'
9A3E   85 3E      STA $3E       flag
9A40   A9 00      LDA #$00      Re-set
9A42   85 20      STA $20       drive status
9A44   F0 DD      BEQ $9A23     Jump to $9A23
9A46³  98         TYA           Repeat drive status
9A47   29 40      AND #$40      Test 'Stepper in operation' flag
9A49   D0 03      BNE $9A4E     Is head moving?
9A4B   4C C9 9A   JMP $9AC9     N—Return from this subroutine
9A4E¹  A5 62      LDA $62       Flag for current stepper phase
9A50   D0 50      BNE $9AA2     Is head in position?
```

```
9A52  A5 4A      LDA $4A         N—Number of steps to be moved
9A54  F0 43      BEQ $9A99       Counter set?
------------------------------------------------------------------
[9A15]
Head control routine
9A56  A5 4A      LDA $4A         YES—Get number of half-steps
9A58  10 59      BPL $9AB3       Should head be moved out?
9A5A  98         TYA             YES—Get drive status and
9A5B  48         PHA             retain it
9A5C  A0 63      LDY #$63        Number of probe attempts (99)
9A5E1 AD 0F 18   LDA $180F       Get ctrl reg. A & set status of
9A61  6A         ROR A           track 0 write-protect in carry
9A62  08         PHP             Save carry
9A63  AD 0F 18   LDA $180F       Read control reg. again and
9A66  6A         ROR A           re-test write-protect notch
9A67  6A         ROR A           Set result in bit 7
9A68  28         PLP             Get previous result
9A69  29 80      AND #$80        Establish last result
9A6B  90 04      BCC $9A71       Is track 0 active in first test?
9A6D  10 1D      BPL $9A8C       N—Is it at track 0 now?
9A6F  30 02      BMI $9A73       YES—Jump to $9A73
9A711 30 19      BMI $9A8C       Is track 0 still active?
------------------------------------------------------------------
Track 0 write-protect notch status remains unchanged
9A731 88         DEY             YES—Try again
9A74  D0 E8      BNE $9A5E       All attempts been performed?
9A76  B0 14      BCS $9A8C       YES—Was track 0 set?
9A78  A5 7B      LDA $7B         YES—Current head control byte set
9A7A  D0 10      BNE $9A8C       by an error?
9A7C  AD 00 1C   LDA $1C00       N—Get drive control register and
9A7F  29 03      AND #$03        get stepper bits
9A81  D0 09      BNE $9A8C       Is a stepper reel active?
9A83  68         PLA             N—repeat drive status
9A84  A8         TAY             and save it
9A85  A9 00      LDA #$00        Clear counter
9A87  85 4A      STA $4A         for steps to be travelled
9A89  4C C9 9A   JMP $9AC9       End
------------------------------------------------------------------
Track 0 write-protect notch status has been changed
9A8C5 68         PLA             repeat drive status and save
9A8D  A8         TAY             it
9A8E  E6 4A      INC $4A         Counter one step out
9A90  AD 00 1C   LDA $1C00       Get control register and
9A93  38         SEC             set stepper bits for
9A94  E9 01      SBC #$01        one step
9A96  4C BB 9A   JMP $9ABB       outward
```

```
9A99[1] A9 02       LDA #$02       Set delay counter to two
9A9B    85 48       STA $48        more IRQs
9A9D    85 62       STA $62        Stepper flag to 'Rest phase'
9A9F    4C C9 9A    JMP $9AC9      Return from this subroutine
9AA2[1] C6 48       DEC $48        Delay for head resting time
9AA4    D0 23       BNE $9AC9      Head ready?
9AA6    A5 20       LDA $20        YES—Get drive status and
9AA8    29 BF       AND #$BF       clear 'Stepper on'
9AAA    85 20       STA $20        flag
9AAC    A9 00       LDA #$00       Set stepper flag
9AAE    85 62       STA $62        back
9AB0    4C C9 9A    JMP $9AC9      Return from this subroutine
------------------------------------------------------------------
[9A58] One half-track step in
9AB3    C6 4A       DEC $4A        Step counter 1 step in
9AB5    AD 00 1C    LDA $1C00      Get control register
9AB8    18          CLC            and set stepper bits
9AB9    69 01       ADC #$01       for one half-track step
------------------------------------------------------------------
[9A96] Set stepper control
9ABB    29 03       AND #$03       inward
9ABD    85 4B       STA $4B        Save value
9ABF    AD 00 1C    LDA $1C00      Get control register
9AC2    29 FC       AND #$FC       and combine new
9AC4    05 4B       ORA $4B        value of
9AC6    8D 00 1C    STA $1C00      stepper bits
9AC9[6] 60          RTS            Return from this subroutine
------------------------------------------------------------------
[9B6C] Format track
9B89    A5 3B       LDA $3B        Get command number and test flags
9B8B    10 03       BPL $9B90      Should track capacity be computed?
9B8D    20 DC 9A    JSR $9ADC      YES—Determine track capacity
9B90[1] AD 26 06    LDA $0626      [Error -- see 7.1.5]
9B93    18          CLC            [Unnecessary operation]
9B94    A9 03       LDA #$03       Set pointers $32/$33
9B96    85 33       STA $33        to beginning
9B98    A9 00       LDA #$00       of
9B9A    85 32       STA $32        data buffer 0
9B9C    8D 28 06    STA $0628      Set first sector number (0)
9B9F    A0 00       LDY #$00       Re-set buffer pointer
9BA1[1] A5 39       LDA $39        Write sector header identifier
9BA3    91 32       STA ($32),Y    in buffer
9BA5    C8          INY            Set buffer pointer to next byte
9BA6    A9 00       LDA #$00       Write empty byte for checksum
9BA8    91 32       STA ($32),Y    in buffer
9BAA    C8          INY            Set buffer pointer to next byte
```

```
9BAB   AD 28 06   LDA $0628      Get sector number and
9BAE   91 32      STA ($32),Y    write in buffer
9BB0   C8         INY            Set buffer pointer to next byte
9BB1   A5 51      LDA $51        Write current track number
9BB3   91 32      STA ($32),Y    in buffer
9BB5   C8         INY            Set buffer pointer to next byte
9BB6   A5 13      LDA $13        Get second ID character and
9BB8   91 32      STA ($32),Y    write in buffer
9BBA   C8         INY            Set buffer pointer to next byte
9BBB   A5 12      LDA $12        Get first ID character and
9BBD   91 32      STA ($32),Y    write in buffer
9BBF   C8         INY            Set buffer pointer to next byte
9BC0   A9 0F      LDA #$0F       Write empty byte value
9BC2   91 32      STA ($32),Y    in buffer
9BC4   C8         INY            Set buffer pointer to next byte
9BC5   91 32      STA ($32),Y    Write to buffer
9BC7   C8         INY            Set buffer pointer to next byte
9BC8   98         TYA            Get buffer pointer
9BC9   48         PHA            and recover it
9BCA   A2 07      LDX #$07       Number of bytes to be included
9BCC   A9 00      LDA #$00       Clear
9BCE   85 3A      STA $3A        checksum
9BD0[1] 88        DEY            Set buffer pointer to prev byte
9BD1   B1 32      LDA ($32),Y    Get byte from header buffer and
9BD3   45 3A      EOR $3A        compute in checksum
9BD5   85 3A      STA $3A        Save value
9BD7   CA         DEX            One more byte
9BD8   D0 F6      BNE $9BD0      Entire header been included?
9BDA   91 32      STA ($32),Y    YES—write checksum in header
9BDC   68         PLA            Reset current buffer
9BDD   A8         TAY            pointer
9BDE   EE 28 06   INC $0628      Go to next sector
9BE1   AD 28 06   LDA $0628      Compare current sector number
9BE4   C5 43      CMP $43        with maximumr number
9BE6   90 B9      BCC $9BA1      Is sector number allowed?
9BE8   A9 03      LDA #$03       NO—Initialize buffer pointer for
9BEA   85 31      STA $31        buffer $0300
9BEC   20 30 FE   JSR $FE30      Convert block header to GCR-bytes
9BEF   A0 BA      LDY #$BA       Turn buff pointer to status buff
9BF1[1] B1 32     LDA ($32),Y    Get byte from status buffer
9BF3   A2 45      LDX #$45       Set pointer to second buffer
9BF5   86 32      STX $32        range
9BF7   91 32      STA ($32),Y    Write GCRbyte in higher buff.area
9BF9   A2 00      LDX #$00       Re-set pointer to
9BFB   86 32      STX $32        beginning
9BFD   88         DEY            Set buffer pointer to next byte
```

```
9BFE   C0 FF      CPY #$FF       Compare with end value
9C00   D0 EF      BNE $9BF1      Copy $300-$344 into $345-$389?
9C02   A0 44      LDY #$44       Buffer pointer to status buffer
9C04¹  B9 BB 01   LDA $01BB,Y    Get byte from status buffer and
9C07   91 32      STA ($32),Y    write to data buffer
9C09   88         DEY            Set buffer pointer to next byte
9C0A   10 F8      BPL $9C04      Entire buffer been transferred?
9C0C   18         CLC            YES—Set buffer pointer to
9C0D   A9 03      LDA #$03       new buffer
9C0F   69 02      ADC #$02       for
9C11   85 31      STA $31        data block contents
9C13   A9 00      LDA #$00       Fill byte value
9C15   A8         TAY            Clear buffer pointer
9C16¹  91 30      STA ($30),Y    Write empty byte in buffer
9C18   C8         INY            Set buffer pointer to next byte
9C19   D0 FB      BNE $9C16      Entire buffer cleared?
9C1B   20 E9 F5   JSR $F5E9      YES—Compute checksum and
9C1E   85 3A      STA $3A        save it
9C20   20 8F F7   JSR $F78F      Convert buffer into GCR-bytes
9C23   A9 00      LDA #$00       Clear pointer to current position
9C25   85 1B      STA $1B        in header buffer
9C27   A2 06      LDX #$06       1536 times $55 (%01010101)
9C29   20 63 9D   JSR $9D63      to diskette
9C2C¹  A0 05      LDY #$05       Number of Sync-bytes
9C2E²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9C31   30 FB      BMI $9C2E      signal
9C33   A9 FF      LDA #$FF       Write byte for Sync-marking
9C35   8D 01 1C   STA $1C01      to diskette
9C38   2C 00 1C   BIT $1C00      Re-set Sync flag
9C3B   88         DEY            Next byte
9C3C   D0 F0      BNE $9C2E      Entire marking written?
9C3E   A2 0A      LDX #$0A       YES—Number of header bytes
9C40   A4 1B      LDY $1B        Get buffer pointer
9C42²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9C45   30 FB      BMI $9C42      signal
9C47   B1 32      LDA ($32),Y    Get byte from header buffer
9C49   8D 01 1C   STA $1C01      and write to diskette
9C4C   2C 00 1C   BIT $1C00      Re-set Sync flag
9C4F   C8         INY            Set buffer pointer to next byte
9C50   CA         DEX            Number of header bytes
9C51   D0 EF      BNE $9C42      Entire header written?
9C53   A0 09      LDY #$09       YES—Number of gap bytes
9C55²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9C58   30 FB      BMI $9C55      signal
9C5A   A9 55      LDA #$55       Write empty byte in gap between
9C5C   8D 01 1C   STA $1C01      header and data block
```

```
9C5F   2C 00 1C   BIT $1C00      Control register reset
9C62   88         DEY            Number of gap bytes
9C63   D0 F0      BNE $9C55      Gap written?
9C65   A9 FF      LDA #$FF       write Sync-marking for
9C67   A0 05      LDY #$05       start of data block
9C69²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9C6C   30 FB      BMI $9C69      signal
9C6E   8D 01 1C   STA $1C01      Write Sync-byte to diskette
9C71   2C 00 1C   BIT $1C00      Initialize input for Sync signal
9C74   88         DEY            Next byte
9C75   D0 F2      BNE $9C69      Is Sync-marking written?
9C77   A0 BB      LDY #$BB       YES—Set buff pntr to status buffr
9C79²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9C7C   30 FB      BMI $9C79      signal
9C7E   B9 00 01   LDA $0100,Y    Get data byte and
9C81   8D 01 1C   STA $1C01      write to diskette
9C84   2C 00 1C   BIT $1C00      Initialize Byte Ready input
9C87   C8         INY            Set buffer pointer to next byte
9C88   D0 EF      BNE $9C79      Entire status buffer written?
9C8A²  2C 0F 18   BIT $180F      YES—Wait for 'Byte ready'
9C8D   30 FB      BMI $9C8A      signal until last byte is written
9C8F   B1 30      LDA ($30),Y    Get byte from data buffer & write
9C91   8D 01 1C   STA $1C01      to diskette
9C94   2C 00 1C   BIT $1C00      Initialize Sync signal input
9C97   C8         INY            Set buffer pointer to next byte
9C98   D0 F0      BNE $9C8A      Data buffer written to diskette?
9C9A   A9 55      LDA #$55       YES—Fillbyte f/gap betwen sectors
9C9C   AC 26 06   LDY $0626      Number of bytes between sectors
9C9F²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9CA2   30 FB      BMI $9C9F      signal
9CA4   8D 01 1C   STA $1C01      Write byte to diskette
9CA7   2C 00 1C   BIT $1C00      Initialize Byte Ready input
9CAA   88         DEY            Next byte
9CAB   D0 F2      BNE $9C9F      Gap written?
9CAD   A5 1B      LDA $1B        YES—Get pointer in header buffer
9CAF   18         CLC            and compute number of GCR-bytes
9CB0   69 0A      ADC #$0A       in header
9CB2   85 1B      STA $1B        Save new pointer
9CB4   CE 28 06   DEC $0628      Decrement number of sectors
9CB7   F0 03      BEQ $9CBC      All sectors written?
9CB9   4C 2C 9C   JMP $9C2C      NO—Write next sector
9CBC²  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9CBF   30 FB      BMI $9CBC      Wait until last byte is written
9CC1   2C 00 1C   BIT $1C00      Initialize Byte Ready input
9CC4¹  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9CC7   30 FB      BMI $9CC4      signal
```

```
9CC9    2C 00 1C   BIT $1C00       Initialize Byte Ready input
9CCC    20 00 FE   JSR $FE00       Switch head for reading
9CCF    A9 C8      LDA #$C8        Determine number of read attempts
9CD1    8D 23 06   STA $0623       (200)
9CD4[1] A9 00      LDA #$00        Clear buffer pointer to current
9CD6    85 1B      STA $1B         header
9CD8    A5 43      LDA $43         Get # of sectors per track and
9CDA    8D 28 06   STA $0628       set in counter
9CDD[1] 20 54 97   JSR $9754       Wait for next Sync-marker
9CE0    A2 0A      LDX #$0A        Number of header bytes
9CE2    A4 1B      LDY $1B         Get pointer to current header and
9CE4[1] B1 32      LDA ($32),Y     get first header byte
9CE6[1] 2C 0F 18   BIT $180F       Wait for 'Byte ready'
9CE9    30 FB      BMI $9CE6       signal
9CEB    CD 01 1C   CMP $1C01       Compare byte from disk w/header
9CEE    D0 0E      BNE $9CFE       Identical?
9CF0    C8         INY             YES—Set pointer to next byte
9CF1    CA         DEX             Number of header bytes
9CF2    D0 F0      BNE $9CE4       Entire header checked?
9CF4    18         CLC             YES—Turn buffer pointer
9CF5    A5 1B      LDA $1B         to next
9CF7    69 0A      ADC #$0A        sector header
9CF9    85 1B      STA $1B         in buffer
9CFB    4C 08 9D   JMP $9D08       Test data block
9CFE[2] CE 23 06   DEC $0623       Decrement number of read attempts
9D01    D0 D1      BNE $9CD4       Any tries left?
9D03    A9 06      LDA #$06        NO—Number for 'Format error'
9D05    4C 51 9D   JMP $9D51       Give return message
9D08[1] 20 54 97   JSR $9754       Wait for next Sync-marking
9D0B    A0 BB      LDY #$BB        Turn buffer pntr to status buffer
9D0D[1] B9 00 01   LDA $0100,Y     get byte from buffer
9D10[1] 2C 0F 18   BIT $180F       Wait for 'Byte ready'
9D13    30 FB      BMI $9D10       signal
9D15    CD 01 1C   CMP $1C01       Compare buffer with diskette
9D18    D0 E4      BNE $9CFE       Identical?
9D1A    C8         INY             YES—Set buffer pnter to next byte
9D1B    D0 F0      BNE $9D0D       Entire buffer examined?
9D1D[1] B1 30      LDA ($30),Y     YES—Get byte from data buffer
9D1F[1] 2C 0F 18   BIT $180F       Wait for 'Byte ready'
9D22    30 FB      BMI $9D1F       signal
9D24    CD 01 1C   CMP $1C01       and compare byte with diskette
9D27    D0 D5      BNE $9CFE       Identical?
9D29    C8         INY             YES—Set buffer pntr to next byte
9D2A    D0 F1      BNE $9D1D       Entire buffer examined?
9D2C    CE 28 06   DEC $0628       YES—Next sector
9D2F    D0 AC      BNE $9CDD       All sectors read?
```

```
9D31   E6 51      INC $51        YES-Set pointer to next track
9D33   A5 51      LDA $51        Get current format track
9D35   2C B1 01   BIT $01B1      Test flag for diskette side
9D38   30 03      BMI $9D3D      Is 2nd side set?
9D3A   C9 24      CMP #$24       NO-Test track for maximum track
9D3C   2C         .byte $2C      Jump next 2 bytes (bit command)
9D3D1  C9 47      CMP #$47       Compare track with max. track(71)
9D3F   B0 03      BCS $9D44      Is current track smaller?
9D41   4C CA 99   JMP $99CA      YES-Move stepper to next track
9D441  A9 FF      LDA #$FF       Clear flag for current
9D46   85 51      STA $51        format track
9D48   A9 00      LDA #$00       Clear flag: 'buffer data in GCR'
9D4A   85 50      STA $50        GCR'
9D4C   A9 01      LDA #$01       Number for 'Ok'
9D4E   4C B5 99   JMP $99B5      Give return message
-------------------------------------------------------------------
[9B70/9D05] end format
9D51   CE 20 06   DEC $0620      Job loop calls still to be called
9D54   F0 03      BEQ $9D59      Call stepper loop again?
9D56   4C CA 99   JMP $99CA      YES-Execute stepper commands
9D592  A0 FF      LDY #$FF       Clear 'Formatting in process'
9D5B   84 51      STY $51        flag
9D5D   C8         INY            Clear 'Buffer data in GCR-Code'
9D5E   84 50      STY $50        flag
9D60   4C B5 99   JMP $99B5      Give return message
-------------------------------------------------------------------
[8D56/9AE0/9C29]
write X times 256 bytes $55 (%01010101) to diskette
9D63   AD 0C 1C   LDA $1C0C      Get control register
9D66   29 1F      AND #$1F       and set head
9D68   09 C0      ORA #$C0       circuitry to
9D6A   8D 0C 1C   STA $1C0C      write mode
9D6D   A9 FF      LDA #$FF       Set head register
9D6F   8D 03 1C   STA $1C03      to output
9D72   A9 55      LDA #$55       Empty byte $55
9D74   A0 00      LDY #$00       Initialize counter
9D763  2C 0F 18   BIT $180F      Wait for 'Byte ready'
9D79   30 FB      BMI $9D76      signal
9D7B   2C 00 1C   BIT $1C00      Initialize input for 'Byte ready'
9D7E   8D 01 1C   STA $1C01      Write byte to diskette
9D81   88         DEY            Decrement counter
9D82   D0 F2      BNE $9D76      256 bytes already written
9D84   CA         DEX            Decrement block counter
9D85   D0 EF      BNE $9D76      Write another 256 bytes?
9D87   60         RTS            NO-Return from this subroutine
-------------------------------------------------------------------
```

```
[thru vector 02A9 from FE67/BF00]
1541 interrupt routine for bus- and disk controller
9D88   48         PHA             Retain accumulator
9D89   8A         TXA             Recover
9D8A   48         PHA             X-register
9D8B   98         TYA             Recover
9D8C   48         PHA             Y-register
9D8D   AD 0D 40   LDA $400D       Get flag for interrupt through
9D90   29 08      AND #$08        serial input/output registers
9D92   F0 26      BEQ $9DBA       Is flag set?
9D94   2C AF 02   BIT $02AF       YES—IRQ mode flag set
9D97   30 21      BMI $9DBA       1571 IRQ routine switched in?
9D99   AD 0F 18   LDA $180F       YES—Switch electronics
9D9C   09 20      ORA #$20        to 1571 mode
9D9E   8D 0F 18   STA $180F       (2 mHz)
9DA1   A9 DE      LDA #$DE        Turn interrupt vector
9DA3   8D A9 02   STA $02A9       in $02A9/$02AA
9DA6   A9 9D      LDA #$9D        to routine
9DA8   8D AA 02   STA $02AA       $99DE
9DAB   A9 40      LDA #$40        Timer 1 (high-byte)
9DAD   8D 07 1C   STA $1C07       set for about 8 ms
9DB0   8D 05 1C   STA $1C05       (2 mHz)
9DB3   A9 00      LDA #$00        Set flag for
9DB5   85 62      STA $62         stepper phase
9DB7   4C EA 9D   JMP $9DEA       1571 job loop
9DBA2  AD 0D 18   LDA $180D       Test interrupt flag
9DBD   29 02      AND #$02        and isolate CA1 input
9DBF   F0 03      BEQ $9DC4       Run into ATN?
9DC1   20 53 E8   JSR $E853       YES—Flags:intrrupt frm serial bus
9DC41  AD 0D 1C   LDA $1C0D       Get interrupt flag register and
9DC7   0A         ASL A           test flag for Timer 1
9DC8   10 03      BPL $9DCD       Timer run?
9DCA   20 B0 F2   JSR $F2B0       YES—Go to 1541 controller routine
9DCD1  BA         TSX             Get stack pointer and
9DCE   BD 04 01   LDA $0104,X     get status from stack
9DD1   29 10      AND #$10        Check flag for jump through 'BRK'
9DD3   F0 03      BEQ $9DD8       Interrupt to be called by 'BRK'?
9DD5   20 B0 F2   JSR $F2B0       YES—execute 1541controler routine
9DD81  68         PLA             Re-set Y-register for
9DD9   A8         TAY             output value
9DDA   68         PLA             Re-set X-register for
9DDB   AA         TAX             output value
9DDC   68         PLA             Get accumulator again
9DDD   40         RTI             Return to break status
-------------------------------------------------------------
```

```
[Over vector 02A9 from FE67/BF03]
1541 interrupt routine for bus- and disk controller
9DDE  48         PHA             Save accumulator
9DDF  8A         TXA             Retain
9DE0  48         PHA             X-register
9DE1  98         TYA             Retain
9DE2  48         PHA             Y-register
9DE3  AD 0D 40   LDA $400D       Get flag for interrupt through
9DE6  29 08      AND #$08        serial i/o registers
9DE8  F0 08      BEQ $9DF2       Is flag set?
9DEA1 A5 37      LDA $37         Get bus status byte and
9DEC  09 40      ORA #$40        set '1571 bus mode'
9DEE  85 37      STA $37         flag
9DF0  D0 22      BNE $9E14       Jump to $9E14
9DF21 AD 0D 18   LDA $180D       test interrupt flags and
9DF5  29 02      AND #$02        isolate CA1 input
9DF7  F0 07      BEQ $9E00       Is ATN found?
9DF9  2C 01 18   BIT $1801       YES-Set flag back
9DFC  A9 01      LDA #$01        Set 'ATN encountered'
9DFE  85 7C      STA $7C         flag
9E001 BA         TSX             Get stack pointer and get
9E01  BD 04 01   LDA $0104,X     status from stack
9E04  29 10      AND #$10        Test 'Jump to BRK' flag
9E06  F0 03      BEQ $9E0B       Will a 'BRK' interrupt be called?
9E08  20 BA 92   JSR $92BA       YES-Execute 1571 jobloop
9E0B1 AD 0D 1C   LDA $1C0D       Get interrupt flag register and
9E0E  0A         ASL A           test Timer 1 flag
9E0F  10 03      BPL $9E14       Timer running?
9E11  20 BA 92   JSR $92BA       YES-Execute 1571 jobloop
9E142 68         PLA             Re-set Y-register for
9E15  A8         TAY             output value
9E16  68         PLA             Re-set Y-register for
9E17  AA         TAX             output value
9E18  68         PLA             Get accumulator again
9E19  40         RTI             Return to break status
-------------------------------------------------------------------
9E1A  FF ...                     unused
9F0C  ... FF                     ROM area
-------------------------------------------------------------------
```

```
Tables for converting 5 GCR-bytes into 4 binary bytes
($FF means that this GCR value is non-existent)
-------------------------------------------------------------------
[9632/9650/993B/9F0F:9683,96F1,9947/9F1D:96A0,994E/9F2A:96C4,995A]
Table for GCR values 2, 4, 5 and 7
9F0D 0C 04 05 FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF
9F1D 0D 0E 80 FF 00 00 10 40 FF 20 C0 60 40 A0 50 E0
9F2D FF FF FF 02 20 08 30 FF FF 00 F0 FF 60 01 70 FF
9F3D FF FF 90 03 A0 0C B0 FF FF 04 D0 FF E0 05 80 FF
9F4D 90 FF 08 0C FF 0F 09 0D 80 02 FF FF FF 03 FF FF
9F5D 00 FF FF 0F FF 0F FF FF 10 06 FF FF FF 07 00 20
9F6D A0 FF FF 06 FF 09 FF FF C0 0A FF FF FF 0B FF FF
9F7D 40 FF FF 07 FF 0D FF FF 50 0E FF FF FF FF 10 30
9F8D B0 FF 00 04 02 06 0A 0E 80 FF FF FF FF FF FF FF
9F9D 20 FF 08 09 80 10 C0 50 30 30 F0 70 90 B0 D0 FF
9FAD FF FF 00 0A FF FF FF FF F0
-------------------------------------------------------------------
[864F/866B/8697/A424/A439/A450/D651/D6D9]
Call job loop and execute job
9FB6  00       BRK           Call job loop
9FB7  EA       NOP           Match jump address
9FB8  B5 00    LDA $00,X     Get job register
9FBA  30 FC    BMI $9FB8     Is job executing?
9FBC  60       RTS           YES—Return from this subroutine
-------------------------------------------------------------------
Table for GCR values 2, 4, 5 and 7  (2nd part)
9FBD 60 FF 01 0B FF FF FF FF 70 FF FF FF FF FF C0 F0
9FCD D0 FF 01 05 03 07 0B FF 90 FF FF FF FF FF FF FF
9FDD A0 FF 0C 0D FF FF FF FF B0 FF FF FF FF FF 40 60
9FED E0 FF 04 0E FF FF FF FF D0 FF FF FF FF FF FF FF
9FFD E0 FF 05 FF FF FF FF FF FF FF FF FF FF FF 50 70
-------------------------------------------------------------------
[9619/9644/9936]  Table for GCR value 1
A00D 0C 04 05 FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF
A01D 0D 0E 80 FF 00 00 10 40 FF 20 C0 60 40 A0 50 E0
A02D FF FF FF 02 20 08 30 30 30 00 F0 FF 60 01 70 FF
A03D FF FF 90 03 A0 0C B0 FF FF 04 D0 FF E0 05 80 FF
A04D 90 FF 08 0C FF 0F 09 0D 80 80 80 80 80 80 80 80
A05D 00 00 00 00 00 00 00 00 10 10 10 10 10 10 10 10
A06D A0 FF FF 06 FF 09 FF FF C0 C0 C0 C0 C0 C0 C0 C0
A07D 40 40 40 40 40 40 40 40 50 50 50 50 50 50 50 50
A08D B0 FF 00 04 02 06 0A 0E 80 80 80 80 80 80 80 80
A09D 20 20 20 20 20 20 20 20 30 30 30 30 30 30 30 30
A0AD FF FF 00 0A 0A 0A 0A 0A F0 F0 F0 F0 F0 F0 F0 F0
A0BD 60 60 60 60 60 60 60 60 70 70 70 70 70 70 70 70
A0CD D0 FF 01 05 03 07 0B FF 90 90 90 90 90 90 90 90
```

```
A0DD A0 A0 A0 A0 A0 A0 A0 A0 B0 B0 B0 B0 B0 B0 B0 B0
A0ED E0 FF 04 0E FF FF FF FF D0 D0 D0 D0 D0 D0 D0 D0
A0FD E0 E0 E0 E0 E0 E0 E0 E0 05 05 05 05 05 05 50 70
---------------------------------------------------------------
[966A/96DA/9942]  Table for GCR-value 3
A10D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A11D FF FF 80 80 00 00 10 10 FF FF C0 C0 40 40 50 50
A12D FF FF FF FF 20 20 30 30 FF FF F0 F0 60 60 70 70
A13D FF FF 90 90 A0 A0 B0 B0 FF FF D0 D0 E0 E0 FF FF
A14D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A15D FF FF 80 80 00 00 10 10 FF FF C0 C0 40 40 50 50
A16D FF FF FF FF 20 20 30 30 FF FF F0 F0 60 60 70 70
A17D FF FF 90 90 A0 A0 B0 B0 FF FF D0 D0 E0 E0 FF FF
A18D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A19D FF FF 80 80 00 00 10 10 FF FF C0 C0 40 40 50 50
A1AD FF FF FF FF 20 20 30 30 FF FF F0 F0 60 60 70 70
A1BD FF FF 90 90 A0 A0 B0 B0 FF FF D0 D0 E0 E0 FF FF
A1CD FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A1DD FF FF 80 80 00 00 10 10 FF FF C0 C0 40 40 50 50
A1ED FF FF FF FF 20 20 30 30 FF FF F0 F0 60 60 70 70
A1FD FF FF 90 90 A0 A0 B0 B0 FF FF D0 D0 E0 E0 FF FF
---------------------------------------------------------------
[96A8/9953]  Table for GCR value 6
A20D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A21D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A22D FF FF FF FF 08 08 08 08 00 00 00 00 01 01 01 01
A23D FF FF FF FF 0C 0C 0C 0C 04 04 04 04 05 05 05 05
A24D FF FF FF FF FF FF FF FF 02 02 02 02 03 03 03 03
A25D FF FF FF FF 0F 0F 0F 0F 06 06 06 06 07 07 07 07
A26D FF FF FF FF 09 09 09 09 0A 0A 0A 0A 0B 0B 0B 0B
A27D FF FF FF FF 0D 0D 0D 0D 0E 0E 0E 0E FF FF FF FF
A28D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A29D FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
A2AD FF FF FF FF 08 08 08 08 00 00 00 00 01 01 01 01
A2BD FF FF FF FF 0C 0C 0C 0C 04 04 04 04 05 05 05 05
A2CD FF FF FF FF FF FF FF FF 02 02 02 02 03 03 03 03
A2DD FF FF FF FF 0F 0F 0F 0F 06 06 06 06 07 07 07 07
A2ED FF FF FF FF 09 09 09 09 0A 0A 0A 0A 0B 0B 0B 0B
A2FD FF FF FF FF 0D 0D 0D 0D 0E 0E 0E 0E FF FF FF FF
---------------------------------------------------------------
[96CC/995F]  Table for GCR value 8
A30D FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A31D FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A32D FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A33D FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A34D FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
```

```
A35D FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A36D FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A37D FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A38D FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A39D FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A3AD FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A3BD FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A3CD FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A3DD FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
A3ED FF FF FF FF FF FF FF FF FF 08 00 01 FF 0C 04 05
A3FD FF FF 02 03 FF 0F 06 07 FF 09 0A 0B FF 0D 0E FF
-----------------------------------------------------------------
[A783/A989]
Format diskette in Commodore 1571 format
A40D   A9 47      LDA #$47        Number of greatest
A40F   8D AC 02   STA $02AC       track to be formatted
A412   A9 03      LDA #$03        Number of current buffer
A414   20 D3 D6   JSR $D6D3       Track/sector to job loop
A417   A2 03      LDX #$03        Go to buffer 3
A419   A9 00      LDA #$00        Set diskette side flag to
A41B   8D B2 01   STA $01B2       side 1
A41E   A9 F0      LDA #$F0        Save 'Format'
A420   85 3B      STA $3B         jobcode
A422   95 00      STA $00,X       Send to job loop
A424   20 B6 9F   JSR $9FB6       Execute job (Format)
A427   C9 02      CMP #$02        Test for 'Ok' message
A429   B0 45      BCS $A470       Jobe run error-free?
A42B   A0 03      LDY #$03        Number of read attempts (4)
A42D1  A9 01      LDA #$01        YES—Send track number (1)
A42F   85 0C      STA $0C         to job loop
A431   A9 00      LDA #$00        Set sector number (0) for
A433   85 0D      STA $0D         job loop
A435   A9 80      LDA #$80        Jobcode for 'Read sector'
A437   95 00      STA $00,X       to job loop
A439   20 B6 9F   JSR $9FB6       Test-read sector 1,0
A43C   C9 02      CMP #$02        Check for 'OK' message
A43E   90 05      BCC $A445       Job run without problems?
A440   88         DEY             Next read attempt
A441   10 EA      BPL $A42D       Have 4 attempts been made?
A443   B0 2B      BCS $A470       YES—Jump to $A470
A4451  A9 01      LDA #$01        Set diskette side flag
A447   8D B2 01   STA $01B2       to side 2
A44A   A9 F0      LDA #$F0        Set 'Format'
A44C   85 3B      STA $3B         jobcode
A44E   95 00      STA $00,X       Give to job loop
A450   20 B6 9F   JSR $9FB6       Execute job
```

```
A453   C9 02       CMP #$02        Compare with 'Ok' message
A455   B0 19       BCS $A470       Job run error-free?
A457   A0 03       LDY #$03        YES—Set number of read attempts
A459¹  A9 24       LDA #$24        Track number (36)
A45B   85 0C       STA $0C         to job loop
A45D   A9 00       LDA #$00        Sector number (0)
A45F   85 0D       STA $0D         to job loop
A461   A9 80       LDA #$80        Set jobcode for 'Read
A463   95 00       STA $00,X       sector'
A465   20 B6 9F    JSR $9FB6       test-read sector 36,0
A468   C9 02       CMP #$02        Check for 'OK'
A46A   B0 01       BCS $A46D       Is there an error?
A46C   60          RTS             NO—Return from this subroutine
A46D¹  88          DEY             Next try
A46E   10 E9       BPL $A459       Three attempts been made?
A470³  A2 00       LDX #$00        YES—Set flag value:'Error noted'
A472   2C 98 02    BIT $0298       in error status flag;
A475   8E 98 02    STX $0298       set new value
A478   10 01       BPL $A47B       Should error be acknowledged?
A47A   60          RTS             NO—Return from this subroutine
A47B¹  4C 0A E6    JMP $E60A       Output error message
-----------------------------------------------------------------
[8294/82A1/885E/BF39]
45-cycle delay
A47E   8A          TXA             Recover X-register
A47F   A2 05       LDX #$05        Set delay value
A481   D0 03       BNE $A486       Jump to $A486
-----------------------------------------------------------------
[8181/8187/8298/82A8/8F0A/8F51/903A/9056/A78E/BF33]
80-cycle delay
A483   8A          TXA             Recover X-register
A484   A2 0D       LDX #$0D        Set delay value
A486¹  CA          DEX             Decrement counter
A487   D0 FD       BNE $A486       End of delay?
A489   AA          TAX             YES—Re-establish X-register
A48A   60          RTS             Return from this subroutine
-----------------------------------------------------------------
[A4C2/A508/A51E/A54F/A5A7/A678/A962]
Recover BAM buffer pointer
A48B   A5 6D       LDA $6D         Get low-byte and
A48D   8D AD 02    STA $02AD       temporarily store
A490   A5 6E       LDA $6E         Get high-byte holen and
A492   8D AE 02    STA $02AE       temporarily store
A495   60          RTS             Return from this subroutine
-----------------------------------------------------------------
```

```
[A4D1/A51B/A531/A58A/A5C2/A6C7/A6E2/A97B]
Re-establish BAM buffer pointer
A496  AD AD 02   LDA $02AD      Get lo-byte from temp. storage &
A499  85 6D      STA $6D        re-set
A49B  AD AE 02   LDA $02AE      Get hi-byte from temp. storage &
A49E  85 6E      STA $6E        re-set
A4A0  60         RTS            Return from this subroutine
-------------------------------------------------------------------
[A851/A887/A8BC/A8F5/A918/A931]
Set pointer to BAM-pattern of sector (for side 2)
A4A1  A6 7F      LDX $7F        Number of current drive (0)
A4A3  BD FF 00   LDA $00FF,X    Get drive status
A4A6  F0 05      BEQ $A4AD      Is drive ready?
A4A8  A9 74      LDA #$74       NO—Give
A4AA  20 48 E6   JSR $E648      '74 drive not ready' error messge
A4AD1 20 19 F1   JSR $F119      Determine channel number of BAM
A4B0  20 DF F0   JSR $F0DF      Read BAM from diskette
A4B3  AD F9 02   LDA $02F9      Test 'legal/illegal BAM' flag
A4B6  F0 07      BEQ $A4BF      Is BAM on disk legal?
A4B8  09 80      ORA #$80       YES—Set 'Write BAM'
A4BA  8D F9 02   STA $02F9      flag
A4BD  D0 03      BNE $A4C2      Jump to $A4C2
A4BF1 20 8D A5   JSR $A58D      Write BAM to diskette
A4C21 20 8B A4   JSR $A48B      Recover BAM buffer pointer
A4C5  20 34 A5   JSR $A534      Set new buffer pointer
A4C8  A5 80      LDA $80        Get number of current track
A4CA  38         SEC            and compute track
A4CB  E9 24      SBC #$24       from side 1,
A4CD  A8         TAY            then get correct # of free blocks
A4CE  B1 6D      LDA ($6D),Y    in track from buffer
A4D0  48         PHA            Save value
A4D1  20 96 A4   JSR $A496      Re-establish old buffer pointer
A4D4  68         PLA            Repeat number of blocks
A4D5  60         RTS            Return from this subroutine
-------------------------------------------------------------------
A4D6  FF ...                    unused
A4E6  ... FF                    ROM area
-------------------------------------------------------------------
```

```
[A854/A88A/A8CF]
Get BAM bit of a sector (for side 2)
A4E7  A5 80     LDA $80       Number of desired track
A4E9  38        SEC           Compute physical track
A4EA  E9 24     SBC #$24      number and
A4EC  A8        TAY           save it
A4ED  A5 81     LDA $81       Get # of desired sector & divide
A4EF  4A        LSR A         by 8 (8 bits per byte)
A4F0  4A        LSR A         and choose corresponding byte of
A4F1  4A        LSR A         three BAM-bytes
A4F2  18        CLC           to position in BAM-pattern
A4F3  79 DB A5  ADC $A5DB,Y   Add track position and save
A4F6  A8        TAY           as pointer to BAM-pattern
A4F7  A5 81     LDA $81       Get number of desired sector and
A4F9  29 07     AND #$07      state position of BAM-bits in
A4FB  AA        TAX           byte-pattern
A4FC  B9 46 01  LDA $0146,Y   Get byte-pattern from BAM-buffer
A4FF  3D E9 EF  AND $EFE9,X   and isolate sector bit
A502  08        PHP           Save value
A503  B9 46 01  LDA $0146,Y   Get entire byte-pattern again and
A506  28        PLP           get previous value from it
A507  60        RTS           Return from this subroutine
-------------------------------------------------------------------
[A862]
Increment number of blocks to a track in BAM
A508  20 8B A4  JSR $A48B     Recover current BAM-pointer
A50B  20 34 A5  JSR $A534     Set pointer to BAM-pattern
A50E  A5 80     LDA $80       Get number of desired track and
A510  38        SEC           calculate physical track
A511  E9 24     SBC #$24      number (side-1 value) and
A513  A8        TAY           save it;
A514  18        CLC           then
A515  B1 6D     LDA ($6D),Y   et byte for number of blocks per
A517  69 01     ADC #$01      track and increment
A519  91 6D     STA ($6D),Y   by one
A51B  4C 96 A4  JMP $A496     Repeat current BAM-pointer
-------------------------------------------------------------------
[A898]
Decrement number of free blocks to a track in BAM
A51E  20 8B A4  JSR $A48B     Recover current BAM-pointer
A521  20 34 A5  JSR $A534     BAMpointer to track's Bytepattern
A524  A5 80     LDA $80       Get number of track desired and
A526  38        SEC           compute physical track number
A527  E9 24     SBC #$24      (Side-1 value) and
A529  A8        TAY           save it
A52A  38        SEC           Decrement number of
```

```
A52B  B1 6D     LDA ($6D),Y     free track blocks
A52D  E9 01     SBC #$01        in appropriate BAM
A52F  91 6D     STA ($6D),Y     byte
A531  4C 96 A4  JMP $A496       Re-set old BAM-pointer
------------------------------------------------------------
[A4C5/A50B/A521/A552/A965]
Set buffer pointer for 2nd buffer from internal channel 6
A534  A2 0D     LDX #$0D        Channel number for 2nd buffer (6)
A536  B5 A7     LDA $A7,X       Get and save pre-arranged
A538  29 0F     AND #$0F        buffer
A53A  AA        TAX             number
A53B  BD E0 FE  LDA $FEE0,X     Get hi-byte of buffer address and
A53E  85 6E     STA $6E         take up in buffer pointer
A540  A9 DD     LDA #$DD        Set low-byte for
A542  85 6D     STA $6D         BAM-pointer
A544  60        RTS             Return from this subroutine
------------------------------------------------------------
[A8BF/A94E]
Verify number of blocks free (side 2 BAM)
A545  A5 6F     LDA $6F         Recover temporary
A547  48        PHA             storage
A548  A5 80     LDA $80         Get current track number (side 2)
A54A  38        SEC             and calculate
A54B  E9 24     SBC #$24        physical number
A54D  A8        TAY             Save
A54E  48        PHA             track number
A54F  20 8B A4  JSR $A48B       Recover current BAM-pointer
A552  20 34 A5  JSR $A534       Set pointer to BAM-pattern
A555  B1 6D     LDA ($6D),Y     Get / save given number of
A557  48        PHA             blocks free
A558  A9 00     LDA #$00        Clear temporary storage area for
A55A  85 6F     STA $6F         number of blocks free
A55C  A9 01     LDA #$01        Set pointer to buffer for
A55E  85 6E     STA $6E         back-side
A560  B9 DB A5  LDA $A5DB,Y     Get pos. of BAM-pattern in buffer
A563  18        CLC             and calculate in buffer area
A564  69 46     ADC #$46        $0146-$01BB
A566  85 6D     STA $6D         Set BAM-pointer
A568  A0 02     LDY #$02        Number of BAM-pattern-bytes -1
A56A1 A2 07     LDX #$07        Number of bits per byte -1
A56C1 B1 6D     LDA ($6D),Y     Get bit-pattern of sectorlayout
A56E  3D E9 EF  AND $EFE9,X     and isolate one bit of sector
A571  F0 02     BEQ $A575       Is sector free?
A573  E6 6F     INC $6F         YES—Increment # of blocks free
A5751 CA        DEX             Test next bit
A576  10 F4     BPL $A56C       Entire byte viewed?
```

```
A578   88         DEY            Include next BAM-byte of track
A579   10 EF      BPL $A56A      All blocks of track checked?
A57B   68         PLA            YES—Get # of BAM blocks given and
A57C   C5 6F      CMP $6F        compare with new number
A57E   F0 05      BEQ $A585      Block layout correct?
A580   A9 71      LDA #$71       NO—Display
A582   20 45 E6   JSR $E645      '71 Dir Error' message
A585¹  68         PLA            Repeat number track
A586   A8         TAY            being worked on
A587   68         PLA            Re-establish
A588   85 6F      STA $6F        temporary storage
A58A   4C 96 A4   JMP $A496      Re-establish BAM-pointer
----------------------------------------------------------------
[A4BF/EF37/F001/F09C]
Write 1571/1541 BAM to diskette
A58D   AD 0F 18   LDA $180F      Get control register and
A590   29 20      AND #$20       get operating mode flag
A592   D0 03      BNE $A597      Is drive in 1541 mode?
A594¹  4C 8A D5   JMP $D58A      YES—Write buffer to disk (1541)
A597¹  AD AC 02   LDA $02AC      Get highest track # from diskette
A59A   C9 25      CMP #$25       and compare with 35
A59C   90 F6      BCC $A594      Diskette two-sided?
A59E   A6 F9      LDX $F9        YES—Get number of current buffer
A5A0   BD 5B 02   LDA $025B,X    Determine and recover
A5A3   48         PHA            appropriate jobcode
A5A4   20 8A D5   JSR $D58A      Write sector to diskette
A5A7   20 8B A4   JSR $A48B      Recover current BAM-pointer
A5AA   20 3A EF   JSR $EF3A      Read BAM into buffer
A5AD   20 08 F0   JSR $F008      Clear buffer for BAM
A5B0   A5 F9      LDA $F9        Get number of current buffer and
A5B2   0A         ASL A          double it (pointers use
A5B3   AA         TAX            2-byte table)
A5B4   A9 35      LDA #$35       Give track 18, side 2(track 53)
A5B6   95 06      STA $06,X      to job loop
A5B8   A0 68      LDY #$68       Buffer pointer to end of 1571 BAM
A5BA¹  B9 46 01   LDA $0146,Y    Get byte from BAM-buffer & write
A5BD   91 6D      STA ($6D),Y    in current data buffer
A5BF   88         DEY            Turn pointer to next byte
A5C0   10 F8      BPL $A5BA      Entire buffer copied?
A5C2   20 96 A4   JSR $A496      YES—re-establish BAM-pointer
A5C5   20 8A D5   JSR $D58A      Write sector to diskette
A5C8   A5 F9      LDA $F9        Get number of current buffer and
A5CA   0A         ASL A          double it (2-byte values for
A5CB   AA         TAX            pointer table)
A5CC   AD 85 FE   LDA $FE85      Give # of directory track(side 1)
A5CF   95 06      STA $06,X      to job loop
```

```
A5D1  20 86 D5   JSR $D586        Read sector from diskette
A5D4  68         PLA              Repeat jobcode
A5D5  A6 F9      LDX $F9          Number of current buffer
A5D7  9D 5B 02   STA $025B,X      Write jobcode again in table
A5DA  60         RTS              Return from this subroutine
-------------------------------------------------------------------
used in : A4F3/A560]
Position of BAM-pattern of track in BAM-buffer
A5DB  00 03 06 09 0C 0F 12 15 18 1B 1E 21 24 27 2A 2D
A5EB  30 33 36 39 3C 3F 42 45 48 4B 4E 51 54 57 5A 5D
A5FB  60 63 66
-------------------------------------------------------------------
[E7A3]
Routine for &-command
A5FE  AD 0F 18   LDA $180F        Get control register and
A601  29 20      AND #$20         test operating mode
A603  F0 0F      BEQ $A614        Is drive in 1571 mode?
A605  A0 00      LDY #$00         YES-[Error -- see 7.1.5]
A607  A2 00      LDX #$00         [unnecessary initialization]
A609  A9 01      LDA #$01         Turn pointer to beginning of
A60B  8D 7A 02   STA $027A        input buffer
A60E  20 12 C3   JSR $C312        Get drive number
A611  4C A8 E7   JMP $E7A8        Return to 1541 &-routine
A614[1] A9 8D    LDA #$8D         Look for endsignal(Shift<Return>)
A616  20 68 C2   JSR $C268        in command string
A619  4C A8 E7   JMP $E7A8        Return to 1541 &-routine
-------------------------------------------------------------------
[EBFC]
Execute command from computer
A61C  20 46 C1   JSR $C146        Execute command string
A61F  20 B2 81   JSR $81B2        Switch 1571 bus to input
A622  A5 37      LDA $37          Get bus status byte and clear
A624  29 7F      AND #$7F         '1571 mode'
A626  85 37      STA $37          flag
A628  4C FF EB   JMP $EBFF        Return to waitloop
-------------------------------------------------------------------
[F997]
Initialize 'motor out' counter
A62B  A9 FF      LDA #$FF         Set counter for the motor
A62D  85 48      STA $48          runtime
A62F  A9 06      LDA #$06         Number of stepper routine calls
A631  85 35      STA $35          yet to be done
A633  60         RTS              Return from this subroutine
-------------------------------------------------------------------
```

```
[F9AB]
Motor on -- wait until turn numbers
A634   D0 07      BNE $A63D      Has 'Write Protect' been changed?
A636   AD AB 02   LDA $02AB      Get high-speed phase counter
A639   D0 10      BNE $A64B      Is motor on turn number?
A63B   F0 1A      BEQ $A657      YES—Jump to $A657
A63D1  A9 FF      LDA #$FF       Set runtime
A63F   8D AB 02   STA $02AB      counter
A642   20 64 87   JSR $8764      Motor on
A645   A9 01      LDA #$01       Set 'Diskette initializing'
A647   85 1C      STA $1C        flag
A649   D0 0C      BNE $A657      Jump to $A657
A64B1  CE AB 02   DEC $02AB      Decrement number of Wait-IRQs
A64E   D0 07      BNE $A657      Is motor on turn number?
A650   A5 20      LDA $20        YES—Get drive status
A652   D0 03      BNE $A657      Motor been on?
A654   20 70 87   JSR $8770      NO—Motor on
A6574  4C B1 F9   JMP $F9B1      Return to head control routine
-------------------------------------------------------------------
[FF15]
Initialize I/O registers
A65A   A9 02      LDA #$02       Set Data output
A65C   8D 00 18   STA $1800      to high
A65F   A9 20      LDA #$20       Switch to 1571mode,turn bus:input
A661   8D 01 18   STA $1801      and head to side 1
A664   4C 18 FF   JMP $FF18      Return to Reset routine
-------------------------------------------------------------------
[D05D/F107]
Read 1571/1541 BAM from diskette
A667   AD 0F 18   LDA $180F      Get control register and test
A66A   29 20      AND #$20       operating mode
A66C   D0 03      BNE $A671      Is drive in 1541 mode?
A66E1  4C 86 D5   JMP $D586      YES—Read sector
A6711  AD AC 02   LDA $02AC      Get highest track number and
A674   C9 25      CMP #$25       compare with 35
A676   90 F6      BCC $A66E      2 sides used?
A678   20 8B A4   JSR $A48B      YES—recover current BAM-pointer
A67B   A9 00      LDA #$00       Turn BAM-pointer to
A67D   85 6D      STA $6D        start-of-buffer
A67F   A6 F9      LDX $F9        Get buffer number and
A681   BD E0 FE   LDA $FEE0,X    get hi-byte of buffer address;
A684   85 6E      STA $6E        set in buffer pointer
A686   A9 FF      LDA #$FF       Set 'Error from job execution
A688   8D 98 02   STA $0298      not noticed'
```

```
A68B   A5 F9      LDA $F9        Repeat buffer number
A68D   0A         ASL A          and double it (table uses 2
A68E   AA         TAX            parameters)
A68F   A9 35      LDA #$35       Give track 18,side 2 (dir. track)
A691   95 06      STA $06,X      to job loop
A693   20 86 D5   JSR $D586      Read sector
A696   C9 02      CMP #$02       Test return message or error
A698   6A         ROR A          Save result in bit 7 (1=error)
A699   29 80      AND #$80       and isolate bit
A69B   49 80      EOR #$80       Prepare bit for tesing in $A6D5
A69D   8D AF 01   STA $01AF      and save it(0= error found)
A6A0   10 0A      BPL $A6AC      Is there an error?
A6A2   A0 68      LDY #$68       NO—Pointer to end of 1571-BAM
A6A4¹  B1 6D      LDA ($6D),Y    Read byte from data buffr & write
A6A6   99 46 01   STA $0146,Y    byte in 1571 BAM-buffer
A6A9   88         DEY            turn pointer to next byte
A6AA   10 F8      BPL $A6A4      All bytes transferred?
A6AC¹  A9 FF      LDA #$FF       Set 'Error by job execution not
A6AE   8D 98 02   STA $0298      notcied' flag
A6B1   A5 F9      LDA $F9        Get number of current data buffer
A6B3   0A         ASL A          and double it (pointers in
A6B4   AA         TAX            2-byte-value table)
A6B5   AD 85 FE   LDA $FE85      Get # of directory track (18) and
A6B8   95 06      STA $06,X      set as track number of job
A6BA   20 86 D5   JSR $D586      Read sector
A6BD   C9 02      CMP #$02       Compare return message w/ 'OK'
A6BF   90 10      BCC $A6D1      Was job run error-free?
A6C1   AA         TAX            YES—Save return message (0/1)
A6C2   A9 24      LDA #$24       Set track 35 as largest track
A6C4   8D AC 02   STA $02AC      (i.e., only one side used)
A6C7   20 96 A4   JSR $A496      Re-establish current BAM-pointer
A6CA   8A         TXA            Repeat return message
A6CB   20 0A E6   JSR $E60A      Output error message
A6CE   4C 44 D6   JMP $D644      Job error handling
A6D1¹  A0 03      LDY #$03       Set buffer pointer,get identifier
A6D3   B1 6D      LDA ($6D),Y    for 1571 diskette ($80)
A6D5   2D AF 01   AND $01AF      Check identfier w/preceding error
A6D8   30 03      BMI $A6DD      Previous read & identifier OK?
A6DA   A9 24      LDA #$24       NO—Use track numbering f/one side
A6DC   2C         .byte $2C      Jump to next 2 bytes(bit command)
A6DD¹  A9 47      LDA #$47       Set track number for 2 diskette
A6DF   8D AC 02   STA $02AC      sides
A6E2   4C 96 A4   JMP $A496      Re-establish BAM-bufferpointer
-----------------------------------------------------------------
```

```
Initialize 1571 diskette
A6E5  20 8C D5   JSR $D58C     Give and run jobcode
A6E8  48         PHA           Save return message
A6E9  C9 02      CMP #$02      Check against 'OK'
A6EB  B0 49      BCS $A736     Job run error-free?
A6ED  AD 0F 18   LDA $180F     YES—Get control register and
A6F0  29 20      AND #$20      determine operating mode
A6F2  F0 42      BEQ $A736     Is drive in 1571 mode?
A6F4  A9 47      LDA #$47      YES—Set max. track number +1 (71)
A6F6  8D AC 02   STA $02AC     --
A6F9  A9 FF      LDA #$FF      Set 'Error from job execution
A6FB  8D 98 02   STA $0298     not noticed' flag
A6FE  A5 16      LDA $16       Recover 1st ID character of
A700  48         PHA           sector header
A701  A5 17      LDA $17       Recover 2nd ID character of
A703  48         PHA           the last sector header
A704  A5 F9      LDA $F9       Get # of current data buffer and
A706  0A         ASL A         double it (table uses
A707  AA         TAX           2-byte-values)
A708  A9 35      LDA #$35      Track 18,side 2(backside direct.)
A70A  95 06      STA $06,X     to job loop
A70C  A9 B0      LDA #$B0      Jobcode for 'Look for sector'
A70E  20 8C D5   JSR $D58C     to job loop; execute
A711  C9 02      CMP #$02      Compare return message with 'OK'
A713  68         PLA           Repeat last character of
A714  A8         TAY           last-read ID
A715  68         PLA           Repeat 1st character of
A716  AA         TAX           last-read ID
A717  B0 0B      BCS $A724     Did job run error-free?
A719  E4 16      CPX $16       YES—Compare last ID with new ID
A71B  D0 07      BNE $A724     Identical?
A71D  C4 17      CPY $17       YES—Compare w/ last ID char. also
A71F  D0 03      BNE $A724     Identical?
A721  A9 47      LDA #$47      YES—# of tracks+1 for 2 sides(71)
A723  2C         .byte $2C     Jump to next 2 bytes(bit command)
A724³ A9 24      LDA #$24      Set # of tracks +1 for 1 side(35)
A726  8D AC 02   STA $02AC     as max. number of tracks
A729  84 17      STY $17       Re-set first-read
A72B  86 16      STX $16       ID
A72D  A5 F9      LDA $F9       Get # of current data buffer
A72F  0A         ASL A         & double it (pointers are
A730  AA         TAX           2-byte-values)
A731  AD 85 FE   LDA $FE85     Get # of directory track (18)
A734  95 06      STA $06,X     Set as track number of job
A736² 68         PLA           Value for 'Ok' return message
A737  60         RTS           Return from this subroutine
```

```
[F005]
Clear 1571 BAM-buffer
A738    20 3A EF   JSR $EF3A      Set buffer pointer
A73B    AD 0F 18   LDA $180F      Get control register
A73E    29 20      AND #$20       and test operating mode
A740    F0 0A      BEQ $A74C      Is drive in 1571 mode?
A742    A9 00      LDA #$00       YES—Value for empty byte
A744    A0 68      LDY #$68       Set buffer pointer
A746[1] 99 46 01   STA $0146,Y    Clear byte in BAM-buffer
A749    88         DEY            Set buffer pointer to next byte
A74A    10 FA      BPL $A746      Entire buffer cleared?
A74C[1] 4C 08 F0   JMP $F008      YES—Set pointer for 1541 BAM
-----------------------------------------------------------------
[F24B]
Compute absolute track number
A74F    48         PHA            Save track number
A750    AD 0F 18   LDA $180F      Get control register
A753    29 20      AND #$20       and test operating mode
A755    F0 08      BEQ $A75F      Is drive in 1571 mode?
A757    68         PLA            YES—Get track # again and compare
A758    C9 24      CMP #$24       with max. number +1 (for side 1)
A75A    90 04      BCC $A760      Is track on side 2?
A75C    E9 23      SBC #$23       YES—Compute track number on side1
A75E    24 68      .byte $24      Jump to next byte (bit-command)
A75F[1] 68         PLA            Re-adjust stack
A760[1] AE D6 FE   LDX $FED6      Get number of track zones
A763    60         RTS            Return from this subroutine
-----------------------------------------------------------------
[EE56]
Create new BAM (1571/1541)
A764    20 05 F0   JSR $F005      Clear BAM-buffer
A767    AD 0F 18   LDA $180F      Get control register
A76A    29 20      AND #$20       and test operaing mode
A76C    D0 03      BNE $A771      Is drive in 1571 mode?
A76E    A9 24      LDA #$24       YES—max. track number +1 (36)
A770    2C         .byte $2C      Jump to next 2 bytes(bit command)
A771[1] A9 47      LDA #$47       Determine max. tracks for 2 sides
A773    8D AC 02   STA $02AC      (71)
A776    4C 43 EE   JMP $EE43      Produce new BAM
-----------------------------------------------------------------
```

```
[FF42]
Format Commodore diskette
A779   AD 0F 18   LDA $180F      Get control register
A77C   29 20      AND #$20       and test operating mode
A77E   D0 03      BNE $A783      Is drive in 1541 mode?
A780   4C C6 C8   JMP $C8C6      YES—Format 1541 diskette
A7831  4C 0D A4   JMP $A40D      Format 1571 diskette
-----------------------------------------------------------------
[EBE4]
Initialize CIA 6526 by reset
A786   AD 01 18   LDA $1801      Get control register
A789   29 DF      AND #$DF       and switch to 1541
A78B   8D 01 18   STA $1801      mode
A78E   20 83 A4   JSR $A483      approx. 80-cycle delay
A791   A9 7F      LDA #$7F       Re-set interrupt
A793   8D 0D 40   STA $400D      register
A796   A9 08      LDA #$08       Set Timer A for 'one shot'
A798   8D 0E 40   STA $400E      (just one run)
A79B   8D 0F 40   STA $400F      Set Timer B for the same mode
A79E   A9 00      LDA #$00       Clear Timer A
A7A0   8D 05 40   STA $4005      high-byte
A7A3   A9 06      LDA #$06       Set Timer A low-byte for 6
A7A5   8D 04 40   STA $4004      cycles
A7A8   A9 01      LDA #$01       Start
A7AA   8D 0E 40   STA $400E      Timer A
A7AD   20 B2 81   JSR $81B2      set 1571 bus for input
A7B0   4C 59 EA   JMP $EA59      test for ATN command mode
-----------------------------------------------------------------
[924B/EA68/EC04]
Take ATN-command from bus
A7B3   AD 0F 18   LDA $180F      Get control register
A7B6   29 20      AND #$20       and test operating mode
A7B8   F0 03      BEQ $A7BD      Is drive in 1571 mode?
A7BA   4C CE 80   JMP $80CE      YES—Get ATN-command from 1571 bus
A7BD1  4C 5B E8   JMP $E85B      Get ATN-command from 1541 bus
-----------------------------------------------------------------
[EB22]
Patch for Reset-routine
A7C0   78         SEI            Disable bus/controller interrupt
A7C1   A2 45      LDX #$45       Initialize stack pointer to
A7C3   9A         TXS            range $100-$145
A7C4   4C 25 EB   JMP $EB25      Return to Reset routine
-----------------------------------------------------------------
```

```
[ED8F/EE56]
Create new 1571/1541 BAM
A7C7  AD 0F 18   LDA $180F      Get control register
A7CA  29 20      AND #$20       and test operating mode
A7CC  D0 09      BNE $A7D7      Is drive in 1541 mode?
A7CE1 A0 03      LDY #$03       Buffer pointr to disktype IDfier
A7D0  A9 00      LDA #$00       Write 1541 diskette identifier
A7D2  91 6D      STA ($6D),Y    in BAM
A7D4  4C B7 EE   JMP $EEB7      Produce 1541 BAM
A7D71 AD AC 02   LDA $02AC      Get largest track number and
A7DA  C9 25      CMP #$25       compare with 37
A7DC  90 F0      BCC $A7CE      Is side 2 used?
A7DE  A0 01      LDY #$01       YES—Determine first track number
A7E0  A2 00      LDX #$00       Set first sector
A7E21 C0 12      CPY #$12       Compare with directory track #
A7E4  F0 34      BEQ $A81A      Directory track already reached?
A7E6  8A         TXA            NO—Save current sector
A7E7  48         PHA            number
A7E8  A9 00      LDA #$00       Clear
A7EA  85 6F      STA $6F        math register for
A7EC  85 70      STA $70        bit-pattern of
A7EE  85 71      STA $71        available sectors
A7F0  B9 2B 94   LDA $942B,Y    Determine and save # of sectors
A7F3  AA         TAX            in track
A7F41 38         SEC            Shift 'Sector free'
A7F5  26 6F      ROL $6F        flag value in math
A7F7  26 70      ROL $70        register for
A7F9  26 71      ROL $71        bit-patterns
A7FB  CA         DEX            Go to next sector
A7FC  D0 F6      BNE $A7F4      All sectors laid out?
A7FE  68         PLA            Re-set first sector number and
A7FF  AA         TAX            re-set buffer pointer
A800  A5 6F      LDA $6F        Get 1st byte from bit-pattern and
A802  9D 46 01   STA $0146,X    write in BAM-buffer
A805  A5 70      LDA $70        Get 2nd byte of bit-pattern and
A807  9D 47 01   STA $0147,X    write in BAM-buffer
A80A  A5 71      LDA $71        Get third byte of bit-pattern and
A80C  9D 48 01   STA $0148,X    write in BAM-buffer
A80F  E8         INX            Jump 3 bytes of bit-pattern
A810  E8         INX            with buffer
A811  E8         INX            pointer
A812  E0 33      CPX #$33       Test for directory track position
A814  D0 04      BNE $A81A      Track 18 already reached?
A816  E8         INX            Jump BAM-entry
A817  E8         INX            from track 18
A818  E8         INX            with buffer pointer
```

```
A819   C8          INY             Set pointer for current track to
A81A2  C8          INY             track 19
A81B   C0 24       CPY #$24        Compare with end of first side
A81D   90 C3       BCC $A7E2       Is track less?
A81F   20 B7 EE    JSR $EEB7       NO—1541 BAM used
A822   A0 03       LDY #$03        Initialize buffer pointer
A824   A9 80       LDA #$80        Write 1571 diskette identifier
A826   91 6D       STA ($6D),Y     in directory sector
A828   A0 FF       LDY #$FF        Set buffer pointer
A82A   A2 22       LDX #$22        # of tracks (w/o directory track)
A82C1  BD 2C 94    LDA $942C,X     Write # of free blocks on track
A82F   91 6D       STA ($6D),Y     in BAM-buffer
A831   88          DEY             Set buffer pointer to next byte
A832   CA          DEX             Pointer to next track entry
A833   10 F7       BPL $A82C       All tracks entered?
A835   A0 EE       LDY #$EE        YES—Turn pointr to track 18,side2
A837   A9 00       LDA #$00        Clear # of free blocks on track
A839   91 6D       STA ($6D),Y     (for directory track)
A83B   4C 75 D0    JMP $D075       Compute free blocks on diskette
---------------------------------------------------------------------
[EF5F]
Free up sector in BAM
A83E   AD 0F 18    LDA $180F       YES—Get control register
A841   29 20       AND #$20        and test operating mode
A843   D0 06       BNE $A84B       Is Floppy in 1541 mode?
A8451  20 CF EF    JSR $EFCF       YES—Set pointr to bit of a sector
A848   4C 62 EF    JMP $EF62       Free up sector
A84B1  A5 80       LDA $80         Get current track number and
A84D   C9 24       CMP #$24        compare with max. value of a side
A84F   90 F4       BCC $A845       Is track number less?
A851   20 A1 A4    JSR $A4A1       NO—Pointer to BAM-bit of sector
A854   20 E7 A4    JSR $A4E7       Get BAM-bit of sector
A857   D0 19       BNE $A872       Is sector free?
A859   1D E9 EF    ORA $EFE9,X     YES—Set BAM-bit
A85C   99 46 01    STA $0146,Y     and write in buffer
A85F   20 88 EF    JSR $EF88       Set 'illegal BAM' flag
A862   20 08 A5    JSR $A508       Increment number of blocks free
A865   A5 80       LDA $80         Test current track number against
A867   C9 35       CMP #$35        track 18, side 2 (Directory)
A869   F0 08       BEQ $A873       Identical?
A86B   A5 7F       LDA $7F         NO—Get current drive number
A86D   0A          ASL A           and double it
A86E   AA          TAX             (table uses 2-byte values)
A86F   4C 7F EF    JMP $EF7F       Increment # of blks free on disk
A8721  38          SEC             'Sector already freed up'errorflg
A8731  60          RTS             Return from this subroutine
```

```
[EF93]
Set sector in BAM
A874    AD 0F 18   LDA $180F     Get control register
A877    29 20      AND #$20      and test operating mode
A879    D0 06      BNE $A881     Is drive in 1541 mode?
A87B[1] 20 CF EF   JSR $EFCF     Set pointer to BAM-bit of sector
A87E    4C 96 EF   JMP $EF96     Free up sector in BAM
A881[1] A5 80      LDA $80       Get # of desired track & test
A883    C9 24      CMP #$24      with max. value +1 for 1st side
A885    90 F4      BCC $A87B     Is track on side 2?
A887    20 A1 A4   JSR $A4A1     YES—Set BAMpointer to track entry
A88A    20 E7 A4   JSR $A4E7     Get BAM-bit of sector
A88D    F0 19      BEQ $A8A8     Is sector freed up?
A88F    5D E9 EF   EOR $EFE9,X   YES—Lay out sector (Bit = 0) and
A892    99 46 01   STA $0146,Y   store BAM pattern again
A895    20 88 EF   JSR $EF88     Set 'Illegal BAM' flag
A898    20 1E A5   JSR $A51E     Get # of blocks free on track
A89B    A5 80      LDA $80       Get # of chosen track and test
A89D    C9 35      CMP #$35      against track 18, side 2
A89F    F0 07      BEQ $A8A8     Identical?
A8A1    A5 7F      LDA $7F       NO—Get current drive number
A8A3    0A         ASL A         and double it
A8A4    AA         TAX           (block table needs 2 bytes)
A8A5    4C B2 EF   JMP $EFB2     Decrement number of blocks free
A8A8[2] 60         RTS           Return from this subroutine
-------------------------------------------------------------
[F1FA]
Look for next free sector on track
A8A9    AD 0F 18   LDA $180F     Get control register
A8AC    29 20      AND #$20      and test operating mode
A8AE    D0 06      BNE $A8B6     Is drive in 1541 mode?
A8B0[1] 20 11 F0   JSR $F011     YES—Set BAM-pointer
A8B3    4C FD F1   JMP $F1FD     Look for next free sector
A8B6[1] A5 80      LDA $80       Check # of current track with
A8B8    C9 24      CMP #$24      max. track +1 of 1st side
A8BA    90 F4      BCC $A8B0     Is track on side 2?
A8BC    20 A1 A4   JSR $A4A1     Set pointer for BAMentry to track
A8BF    20 45 A5   JSR $A545     Check # of blocks free on track
A8C2    B9 2C 94   LDA $942C,Y   Get # of sectors per track and
A8C5    8D 4E 02   STA $024E     save it
A8C8[1] A5 81      LDA $81       Compare number of current sector
A8CA    CD 4E 02   CMP $024E     with max. sector number
A8CD    B0 09      BCS $A8D8     Is sector # in allowable range?
```

```
A8CF   20 E7 A4   JSR $A4E7        YES-Get BAM-bit of sector
A8D2   D0 06      BNE $A8DA        Is sector free?
A8D4   E6 81      INC $81          NO-Go to next sector
A8D6   D0 F0      BNE $A8C8        Jump to $A8C8
A8D8¹  A9 00      LDA #$00         Flag for 'No free sector'
A8DA¹  60         RTS              Return from this subroutine
---------------------------------------------------------------
[F12D] Look for next free sector
A8DB   AD 0F 18   LDA $180F        Get control register
A8DE   29 20      AND #$20         and determine operating mode
A8E0   D0 06      BNE $A8E8        Is disk in 1541 mode?
A8E2¹  A5 6F      LDA $6F          YES-Recover number of free blocks
A8E4   48         PHA              per track
A8E5   4C 30 F1   JMP $F130        Look for next free sector
A8E8¹  A5 80      LDA $80          Test current track number against
A8EA   C9 24      CMP #$24         max. track +1 of side 1
A8EC   90 F4      BCC $A8E2        Is track on side 2?
A8EE   C9 35      CMP #$35         YES-test for track 18, side 2
A8F0   F0 0E      BEQ $A900        Identical?
A8F2   A5 6F      LDA $6F          Recover
A8F4   48         PHA              zeropage area
A8F5   20 A1 A4   JSR $A4A1        Set BAM-pattern pointer
A8F8   A8         TAY              Save number of free blocks
A8F9   68         PLA              Re-establish
A8FA   85 6F      STA $6F          zeropage area
A8FC   98         TYA              Get # of blocks free on track
A8FD   4C 38 F1   JMP $F138        Get free sector
A900¹  A9 00      LDA #$00         Set # of free blocks on track
A902   4C 38 F1   JMP $F138        Look for next free sector
---------------------------------------------------------------
[F1C4]
Set BAM pointer to bit on a sector
A905   AD 0F 18   LDA $180F        Get control register
A908   29 20      AND #$20         and determine operating mode
A90A   D0 06      BNE $A912        Is disk in 1541 mode?
A90C¹  20 11 F0   JSR $F011        YES-Set BAM-pointer and
A90F   4C C7 F1   JMP $F1C7        get optimal free sector
A912¹  A5 80      LDA $80          Test current track number against
A914   C9 24      CMP #$24         max. track +1 of side 1
A916   90 F4      BCC $A90C        Is track on side 2?
A918   20 A1 A4   JSR $A4A1        Set BAM-pointer
A91B   4C C9 F1   JMP $F1C9        Get optimal free sector
---------------------------------------------------------------
```

```
[F1D1]
Look for free sector
A91E  AD 0F 18   LDA $180F       Get control register
A921  29 20      AND #$20        and determine operating mode
A923  D0 06      BNE $A92B       Is disk in 1541 mode?
A925¹ 20 11 F0   JSR $F011       YES—Set BAM-pointer
A928  4C E2 F1   JMP $F1E2       Look for free sector
A92B¹ A5 80      LDA $80         Test current track number against
A92D  C9 24      CMP #$24        max. track +1 of side 1
A92F  90 F4      BCC $A925       Is track on side 2?
A931  20 A1 A4   JSR $A4A1       Set pointer to BAM-bit
A934  4C E4 F1   JMP $F1E4       Look for free sector
--------------------------------------------------------------------
[EF28]
Test number of free blocks in BAM
A937  AD 0F 18   LDA $180F       Get control register
A93A  29 20      AND #$20        and determine operating mode
A93C  D0 03      BNE $A941       Is disk in 1541 mode?
A93E² 4C 20 F2   JMP $F220       YES—Test block assignment
A941¹ AD AC 02   LDA $02AC       Get number of largest track;
A944  C9 25      CMP #$25        compare with 37
A946  90 F6      BCC $A93E       Only one diskette used?
A948  A5 80      LDA $80         NO—Get current track # and
A94A  C9 24      CMP #$24        compare with 36
A94C  90 F0      BCC $A93E       Is track on side 2?
A94E  4C 45 A5   JMP $A545       YES—Confirm block assignment
--------------------------------------------------------------------
[D097]
Compute number of free blocks on diskette
A951  9D FA 02   STA $02FA,X     Store low-byte of free blocks
A954  AD 0F 18   LDA $180F       Get control register
A957  29 20      AND #$20        and determine operating mode
A959  F0 23      BEQ $A97E       Is disk in 1571 mode?
A95B  AD AC 02   LDA $02AC       YES—Determine maximum track #
A95E  C9 25      CMP #$25        & compare w/ maximum track+2 (37)
A960  90 1C      BCC $A97E       Is track on side 2?
A962  20 8B A4   JSR $A48B       Recover current BAM-pointer
A965  20 34 A5   JSR $A534       Set BAM-pointer to track entry
A968  A0 22      LDY #$22        Number of tracks on a side -1
A96A  AD FA 02   LDA $02FA       Get low-byte of free block and
A96D¹ 18         CLC             include byte of free blocks on
A96E  71 6D      ADC ($6D),Y     track
A970  8D FA 02   STA $02FA       Save next block amount
A973  90 03      BCC $A978       Is a transfer pending?
```

```
A975   EE FC 02   INC $02FC       YES—Set hi-byte of 'Blocks free'
A978¹  88         DEY             Go to next track
A979   10 F2      BPL $A96D       All tracks considered?
A97B   4C 96 A4   JMP $A496       YES—Set BAM-pointer to old value
A97E²  60         RTS             Return from this subroutine
----------------------------------------------------------------

[DCD6]
Patch for number of free blocks; Clear pointer
A97F   95 B5      STA $B5,X       Clear # of reserved blocks / file
A981   95 BB      STA $BB,X       (low- and high-byte)
A983   A9 00      LDA #$00        Clear number of data bytes yet to
A985   9D 44 02   STA $0244,X     be transferred
A988   60         RTS             Return from this subroutine
----------------------------------------------------------------

[84E4]
Format diskette im 1571 format
A989   20 0D A4   JSR $A40D       Format diskette
A98C   A0 00      LDY #$00        'Error noted'
A98E   8C 98 02   STY $0298       flag
A991   60         RTS             Return from this subroutine
----------------------------------------------------------------
A992   FF ...                     unused
A99C   ... FF                     ROM area
----------------------------------------------------------------
[C1B3]  Patch for 1541 routine (Error remedied by FF,X)
A99D   A9 00      LDA #$00        Set drive status for
A99F   9D FF 00   STA $00FF,X     'Drive  ready'
A9A2   4C B7 C1   JMP $C1B7       Return to 1541 routine
----------------------------------------------------------------
[C661]  Patch for 1541 routine (Error remedied by FF,X)
A9A5   98         TYA             Get return message
A9A6   9D FF 00   STA $00FF,X     and transfer into drive status
A9A9   4C 64 C6   JMP $C664       Return to 1541 routine
----------------------------------------------------------------
[EA6B]
Work with serial bus after ATN-command
A9AC   AD 0F 18   LDA $180F       Get control register
A9AF   29 20      AND #$20        test for operating mode
A9B1   F0 03      BEQ $A9B6       Is disk in 1571 mode?
A9B3   4C 5A 81   JMP $815A       YES—Wait on 1571 bus
A9B6¹  4C D7 E8   JMP $E8D7       Process 1541 bus
----------------------------------------------------------------
```

```
[E60A]
Display error message and prepare text version of message
A9B9  48          PHA          Recover error number
A9BA  86 F9       STX $F9      Save buffer number
A9BC  AD 0F 18    LDA $180F    Get control register
A9BF  29 20       AND #$20     and test for operating mode
A9C1  F0 0F       BEQ $A9D2    Is disk in 1571 mode?
A9C3  24 37       BIT $37      YES—Chk bus statusbyte f/1571mode
A9C5  10 0B       BPL $A9D2    Flag set?
A9C7  A5 37       LDA $37      YES—Clear 1571 mode
A9C9  29 7F       AND #$7F     flag in bus status
A9CB  85 37       STA $37      byte
A9CD  68          PLA          Repeat error number and
A9CE  AA          TAX          prepare for output
A9CF  4C 99 91    JMP $9199    Produce #, message over 1571 bus
A9D2² 4C 0D E6    JMP $E60D    Prepare text of message
-----------------------------------------------------------------
[C1CE]
Produce error message in error buffer
A9D5  48          PHA          Recover error number
A9D6  AD 0F 18    LDA $180F    Get control register
A9D9  29 20       AND #$20     and test for operating mode
A9DB  F0 17       BEQ $A9F4    Is disk in 1571 mode?
A9DD  24 37       BIT $37      YES—Test bus statsbyte f/1571mode
A9DF  10 13       BPL $A9F4    Flag set?
A9E1  A5 37       LDA $37      YES—Clear 1571 mode flag
A9E3  29 7F       AND #$7F     in bus status
A9E5  85 37       STA $37      byte
A9E7  78          SEI          Disable bus/controller interrupt
A9E8  A2 02       LDX #$02     Send 'File Not Found' error #
A9EA  20 28 92    JSR $9228    over 1571 bus
A9ED  A9 00       LDA #$00     Set secondary address
A9EF  85 83       STA $83      for Load
A9F1  20 C0 DA    JSR $DAC0    Close file
A9F4² 68          PLA          Repeat error number
A9F5  4C 45 E6    JMP $E645    Produce error message
-----------------------------------------------------------------
[F263]  Patch for 1541 routine (new: Clear status)
A9F8  A9 00       LDA #$00     Clear status for drive
A9FA  85 20       STA $20      0
A9FC  AD 0C 1C    LDA $1C0C    Get peripheral control register
A9FF  4C 66 F2    JMP $F266    Return to 1541 routine
-----------------------------------------------------------------
```

```
[C2BA/C2C2]
Observe new User0 command
AA02   AD 00 02   LDA $0200      Get 1st char from command string
AA05   C9 55      CMP #$55       & compare with 'U' (User command)
AA07   D0 07      BNE $AA10      Identical?
AA09   AD 01 02   LDA $0201      YES—Get 2nd char frm cmd string
AA0C   C9 30      CMP #$30       and compare with '0'
AA0E   F0 04      BEQ $AA14      Is command 'U0'?
AA10¹  B9 00 02   LDA $0200,Y    NO—Get char from command string
AA13   2C         .byte $2C      Jump next 2 bytes (bit command)
AA14¹  A9 00      LDA #$00       Give back empty params by User 0
AA16   60         RTS            Return from this subroutine
---------------------------------------------------------------------
[C66B]  Patch for 1541 routine (Error remedied by FF,X)
AA17   A6 7F      LDX $7F        Current drive number
AA19   BD FF 00   LDA $00FF,X    Get appropriate drive status
AA1C   60         RTS            Return from this subroutine
---------------------------------------------------------------------
[D071]  Patch for 1541 routine (Error remedied by FF,X)
AA1D   95 1C      STA $1C,X      Set diskette initialization flag
AA1F   9D FF 00   STA $00FF,X    Set drive status
AA22   4C 75 D0   JMP $D075      Return to 1541 routine
---------------------------------------------------------------------
[F017]  Patch for 1541 routine (Error remedied by FF,X)
AA25   A6 7F      LDX $7F        Current drive number
AA27   BD FF 00   LDA $00FF,X    Get appropriate drive status
AA2A¹  4C 1B F0   JMP $F01B      Return to 1541 routine
---------------------------------------------------------------------
[CB81]
Execute User-command (UA to UK)
AA2D   A5 75      LDA $75        Get low-byte of User-routine and
AA2F   C9 67      CMP #$67       test with IRQ
AA31   D0 09      BNE $AA3C      Identical?
AA33   A5 76      LDA $76        YES—Get high-byte and compare
AA35   C9 FE      CMP #$FE       with IRQ address
AA37   D0 03      BNE $AA3C      Identical?
AA39   00         BRK            YES—Call jobloop
AA3A   EA         NOP            Cancel out return address
AA3B   60         RTS            Return from this subroutine
AA3C²  6C 75 00   JMP ($0075)    Execute User-command
---------------------------------------------------------------------
AA3F   FF ...                    unused
BEFF   ... FF                    ROM area
---------------------------------------------------------------------
```

```
[Table not used by DOS]
Table of most important DOS routines
BF00  4C 88 9D   JMP $9D88        1541 IRQ routine
BF03  4C DE 9D   JMP $9DDE        1571 IRQ routine
BF06  4C B0 F2   JMP $F2B0        1541 jobloop
BF09  4C BA 92   JMP $92BA        1571 jobloop
BF0C  4C 93 F3   JMP $F393        Set buffer pointer for Jobloop
BF0F  4C D1 93   JMP $93D1        Set buffer pointer for Jobloop
BF12  4C 69 F9   JMP $F969        Conclude Job; Give return message
BF15  4C B5 99   JMP $99B5        Conclude Job; Give return message
BF18  4C 00 FE   JMP $FE00        Switch head to read mode
BF1B  4C 34 F9   JMP $F934        Convert block header to GCRvalues
BF1E  4C 56 F5   JMP $F556        Wait for Sync-marking (1541)
BF21  4C 54 97   JMP $9754        Wait for Sync-marking (1571)
BF24  4C E0 F8   JMP $F8E0        Convrt statsbuff from GCR to bin.
BF27  4C 65 99   JMP $9965        Convrt statsbuff from GCR to bin.
BF2A  4C E9 F5   JMP $F5E9        Compute sector checksum
BF2D  4C E6 F7   JMP $F7E6        Convert 5 GCRbytes to 4 bin.bytes
BF30  4C D9 98   JMP $98D9        Convert 5 GCRbytes to 4 bin.bytes
BF33  4C 83 A4   JMP $A483        Wait approx. 80 cycles
BF36  4C F3 FE   JMP $FEF3        Delay for 1541 serial bus
BF39  4C 7E A4   JMP $A47E        Wait approx. 45 cycles
BF3C  4C 05 F0   JMP $F005        Clear buffer for BAM
BF3F  4C D1 F0   JMP $F0D1        Get track number for BAM
BF42  4C 46 C1   JMP $C146        Execute command string
BF45  4C 68 C2   JMP $C268        Search cmd string f/paramaters
BF48  4C B3 C2   JMP $C2B3        Set pnter for cmd string analyses
BF4B  4C DC C2   JMP $C2DC        Clear all file pointers
BF4E  4C E6 86   JMP $86E6        Execute routine w/#in accumulator
BF51  4C 64 87   JMP $8764        Drive motor on
BF54  4C 70 87   JMP $8770        Drive motor off
BF57  4C 8E 80   JMP $808E        [Error -- see 7.1.5]
BF5A  4C 1E CF   JMP $CF1E        Look for and set buffer
BF5D  4C B4 D7   JMP $D7B4        Execute Open command from bus
BF60  4C C0 DA   JMP $DAC0        Close channel and close file
BF63  4C 0A E6   JMP $E60A        Send eror message from job loop
BF66  4C 80 90   JMP $9080        Read file (PRG/SEQ/USR)
BF69  4C 4E 92   JMP $924E        Test ROM checksum
BF6C  4C 59 F2   JMP $F259        Initialize 1541 disk controller
BF6F  4C 9C F9   JMP $F99C        1541 job loop off
BF72  4C CA 99   JMP $99CA        Stepper control
BF75  4C 95 FE   JMP $FE95        [Error -- see 7.1.5]
--------------------------------------------------------------------
BF78  FF ...                      unused
C0FF  ... FF                      ROM area
--------------------------------------------------------------------
```

```
[C38C/C439/C46A/C49A/CE0B/CFA2/D39E/D7E1]  cf. 877C/C118
LED on current drive ON
(Routine taken from old double drives)
C100  78        SEI         Disable bus/controller interrupt
C101  A9 F7     LDA #$F7    Place mask for LED-bit (Bit3)
C103  2D 00 1C  AND $1C00   so that LED-bit will be cleared
C106  48        PHA         Save mask
C107  A5 7F     LDA $7F     Drive number (always 0)
C109  F0 05     BEQ $C110   Otherwise, jump to $C110
C10B  68        PLA         Unused if system has only one
C10C  09 00     ORA #$00    drive
C10E  D0 03     BNE $C113   If drive 1 exists, jump to $C113
C110¹ 68        PLA         Repeat mask
C111  09 08     ORA #$08    LED-bit set (Bit3=1)
C113¹ 8D 00 1C  STA $1C00   LED on
C116  58        CLI         Enable bus/controller interrupt
C117  60        RTS         Return from this subroutine
-----------------------------------------------------------
[Routine not used in DOS]  cf. 877C/C100
LED on 1571 dive ON
C118  78        SEI         Disable bus/controller interrupt
C119  A9 08     LDA #$08    LED-Bit (Bit3) set (Bit3=1)
C11B  0D 00 1C  ORA $1C00   Take up other bits of register
C11E  8D 00 1C  STA $1C00   LED on
C121  58        CLI         Enable bus/controller interrupt
C122  60        RTS         Return from this subroutine
-----------------------------------------------------------
[C1AA/D425/E6BC]
Clear error flag
C123  A9 00     LDA #$00    Clear the error flag
C125  8D 6C 02  STA $026C   Clear error number
C128  8D 6D 02  STA $026D   Clear LED-Blinker flag
C12B  60        RTS         Return from this subroutine
-----------------------------------------------------------
[E650]
LED blinker on due to error
C12C  78        SEI         Disable bus/controller interrupt
C12D  8A        TXA         Recover
C12E  48        PHA         X-register
C12F  A9 50     LDA #$50    Set LED blink counter
C131  8D 6C 02  STA $026C   to 80
C134  A2 00     LDX #$00    Go to drive 0
C136  BD CA FE  LDA $FECA,X Save LED-mask for chosen
C139  8D 6D 02  STA $026D   drive
C13C  0D 00 1C  ORA $1C00   Drive LED
C13F  8D 00 1C  STA $1C00   on
```

```
C142  68          PLA               Reset
C143  AA          TAX               X-register
C144  58          CLI               Enable bus/controller interrupt
C145  60          RTS               Return from this subroutine
-------------------------------------------------------------------
[A61C/BF42]
Execute command string from computer
C146  A9 00       LDA #$00          Set 'Write BAM to diskette'
C148  8D F9 02    STA $02F9         flag
C14B  AD 8E 02    LDA $028E         Take on last-used drive as
C14E  85 7F       STA $7F           current dive
C150  20 BC E6    JSR $E6BC         Produce 'OK' message
C153  A5 84       LDA $84           Get last IEC secondary address
C155  10 09       BPL $C160         Was it a Close command?
C157  29 0F       AND #$0F          Get number of chosen channel and
C159  C9 0F       CMP #$0F          test for command channel
C15B  F0 03       BEQ $C160         Is command channel being used?
C15D  4C B4 D7    JMP $D7B4         NO—
C160² 20 B3 C2    JSR $C2B3         Set params to command processing
C163  B1 A3       LDA ($A3),Y       Get/save 1st char from
C165  8D 75 02    STA $0275         input buffer
C168  A2 0B       LDX #$0B          Number of disk commands
C16A¹ BD 89 FE    LDA $FE89,X       Get char from 1541command table &
C16D  CD 75 02    CMP $0275         compare with command characters
C170  F0 08       BEQ $C17A         Is that the desired command?
C172  CA          DEX               NO—Go to next command character
C173  10 F5       BPL $C16A         Compared with all commands?
C175  A9 31       LDA #$31          YES—Display
C177  4C C8 C1    JMP $C1C8         '31 Syntax Error' message
C17A¹ 8E 2A 02    STX $022A         Save command number
C17D  E0 09       CPX #$09          Compare with number for 'Rename'
C17F  90 03       BCC $C184         Is command 'R', 'S' or 'N' ?
C181  20 EE C1    JSR $C1EE         YES—Check syntax
C184¹ AE 2A 02    LDX $022A         Get back command number
C187  BD 95 FE    LDA $FE95,X       Get lo-byte of command of table &
C18A  85 6F       STA $6F           set in pointer
C18C  BD A1 FE    LDA $FEA1,X       Transfer hi-byte of start address
C18F  85 70       STA $70           --
C191  6C 6F 00    JMP ($006F)       Execute command
-------------------------------------------------------------------
```

```
[9196/C99E/C9A4/CAC9/CB6F/CC18/CCFB/CD16/CD5C/CD70/CD94/CDA0/CDB7/CDCF]
[D00B/D7F0/D99D/DAE6/DAFC/E272/EDB0/EEB4]
End of computer command; produce an error message
C194  A9 00     LDA #$00        Clear Write BAM to diskette'
C196  8D F9 02  STA $02F9       flag
------------------------------------------------------------------
[DA06]  End of command;but don't write BAM to diskette
C199  AD 6C 02  LDA $026C       Get error flag
C19C  D0 2A     BNE $C1C8       Is an error extant?
C19E  A0 00     LDY #$00        NO—Prepare OK-message
C1A0  98        TYA             Clear error number
------------------------------------------------------------------
[C87A]  end of command; ignore error
C1A1  84 80     STY $80         Set track and
C1A3  84 81     STY $81         sector to zero
C1A5  84 A3     STY $A3         Set back input buffer pointer
C1A7  20 C7 E6  JSR $E6C7       Produce 'OK'-message
C1AA  20 23 C1  JSR $C123       Clear error flags
------------------------------------------------------------------
[DAE9/DAFF]  End of command; no return message prepared
C1AD  A5 7F     LDA $7F         Save current drive (usually 0)
C1AF  8D 8E 02  STA $028E       as last drive number
C1B2  AA        TAX             for current drive
C1B3  4C 9D A9  JMP $A99D       Clear 'Drive active' flag
C1B6  EA        NOP             [through modification of 1541ROM]
C1B7  20 BD C1  JSR $C1BD       Clear input buffer ($0200-$0228)
C1BA  4C DA D4  JMP $D4DA       Close internal read/write chanels
------------------------------------------------------------------
[C1B7/E648]
Clear input buffer for command from computer
C1BD  A0 28     LDY #$28        Overwrite 41 character positions
C1BF  A9 00     LDA #$00        with zero
C1C1[1] 99 00 02  STA $0200,Y   (range $0200-$0228)
C1C4  88        DEY             Next character
C1C5  10 FA     BPL $C1C1       All characters cleared yet?
C1C7  60        RTS             YES—Return from this subroutine
------------------------------------------------------------------
```

```
[850E/85AC/9023/91C7/C177/C19C/C1F5/C265/C2D9/C41D/C8C3/C925/C984/CAE3]
[CAF1/CB4D/CBA2/CC28/CC2D/CD33/CDE2/CF78/CFFA/D214/D38E/D839/D8ED/D8F2]
[D947/D959/D967/D9C0/E0D3/E16B/E216/E225/E299/E365/E44B/E7C2/EE16/F0EF]
[F15C]
Display error message and necessary number in accumulator
C1C8  A0 00     LDY #$00     Clear pointer
C1CA  84 80     STY $80      Clear track number and
C1CC  84 81     STY $81      sector number
C1CE  4C D5 A9  JMP $A9D5    Put message in buffer
--------------------------------------------------------------
[D005/D7FF/ED84]
Look for ':' and drive number in command string
(Y-Register must point to current position in buffer)
C1D1  A2 00     LDX #$00     Clear pointer to drive# position
C1D3  8E 7A 02  STX $027A    in input buffer
C1D6  A9 3A     LDA #$3A     Look for a colon when looking for
C1D8  20 68 C2  JSR $C268    characters in input buffer
C1DB  F0 05     BEQ $C1E2    Has colon been found?
C1DD  88        DEY          YES—Y-Register shows position+1
C1DE  88        DEY          of character;
C1DF  8C 7A 02  STY $027A    Drive # position (before ':')
C1E21 4C 68 C3  JMP $C368    Set drive and turn LED on
--------------------------------------------------------------
[C1EE/C904/D82B/DA86]
Look for colon in command string
C1E5  A0 00     LDY #$00     Starting position of search
C1E7  A2 00     LDX #$00     Number of filenames found
C1E9  A9 3A     LDA #$3A     Colon declared as sought char
C1EB  4C 68 C2  JMP $C268    Search through input buffer
--------------------------------------------------------------
[C181]
Test command with two filenames for syntax
C1EE  20 E5 C1  JSR $C1E5    Look for colon in input buffer
C1F1  D0 05     BNE $C1F8    Has colon been found?
C1F31 A9 34     LDA #$34     NO—Display
C1F5  4C C8 C1  JMP $C1C8    '34 Syntax Error' error
C1F82 88        DEY          Y-Register points to position +1
C1F9  88        DEY          Set pointer before ':'
C1FA  8C 7A 02  STY $027A    Save position of drive number
C1FD  8A        TXA          Number of filenames already found
C1FE  D0 F3     BNE $C1F3    Have several names been found?
C2001 A9 3D     LDA #$3D     NO—'=' - character
C202  20 68 C2  JSR $C268    Look at line after '='
C205  8A        TXA          Number of filenames found so far
C206  F0 02     BEQ $C20A    Has only one file been found?
C208  A9 40     LDA #$40     NO—Bit 6 as flag for more files
```

```
C20A1 09 21     ORA #$21     Bit 0 & 5 for'1st filename found'
C20C  8D 8B 02  STA $028B    Save bitflags
C20F  E8        INX          Point to end of 1st filename
C210  8E 77 02  STX $0277    Number of files found by 1st
C213  8E 78 02  STX $0278    file description
C216  AD 8A 02  LDA $028A    Wildcard flag ('*')
C219  F0 0D     BEQ $C228    Wildcard onhand in filename?
C21B  A9 80     LDA #$80     YES—Set flag in syntax byte
C21D  0D 8B 02  ORA $028B    and
C220  8D 8B 02  STA $028B    save flag again
C223  A9 00     LDA #$00     Clr search routine wildcard flag
C225  8D 8A 02  STA $028A    --
C2281 98        TYA          Position of '=' - char in command
C229  F0 29     BEQ $C254    End of command line found?
C22B  9D 7A 02  STA $027A,X  NO—Save position of filename
C22E  AD 77 02  LDA $0277    The # of files for 1st filenaming
C231  8D 79 02  STA $0279    set as number for second naming
C234  A9 8D     LDA #$8D     Look for command string endmarker
C236  20 68 C2  JSR $C268    and continue search to end
C239  E8        INX          Save # of commas found;from that
C23A  8E 78 02  STX $0278    save the # of additionl filenames
C23D  CA        DEX          Establish original value
C23E  AD 8A 02  LDA $028A    Wildcard flag ('*')
C241  F0 02     BEQ $C245    Wildcard onhand?
C243  A9 08     LDA #$08     YES—Set Bit 3 as flag
C2451 EC 77 02  CPX $0277    Compare length with old value
C248  F0 02     BEQ $C24C    Any more filenames found?
C24A  09 04     ORA #$04     YES—Flag:filenames aftr '=' char.
C24C1 09 03     ORA #$03     Flag for '=' character onhand
C24E  4D 8B 02  EOR $028B    Combine previous flags & save as
C251  8D 8B 02  STA $028B    new syntax status
C2541 AD 8B 02  LDA $028B    Syntax flag for command
C257  AE 2A 02  LDX $022A    Compare onhand command numbers
C25A  3D A5 FE  AND $FEA5,X  with allowable syntax;
C25D  D0 01     BNE $C260    all legal?
C25F  60        RTS          YES—Return from this subroutine
C2601 8D 6C 02  STA $026C    Save type of incorrect syntax
C263  A9 30     LDA #$30     Display
C265  4C C8 C1  JMP $C1C8    '30 Syntax Error'
---------------------------------------------------------------------
```

```
[C1D8/C1EB/C202/C236/CC21/CC75/D845]
Search input line for character from accumulator
(Y-Register must contain current position in input buffer;)
(X-Register contains number of parameters found)
C268   8D 75 02   STA $0275        Chars looked for by system
C26B2  CC 74 02   CPY $0274        Test length of command string
C26E   B0 2E      BCS $C29E        End reached?
C270   B1 A3      LDA ($A3),Y      NO—Get char from input buffer
C272   C8         INY              Set pointer to next character
C273   CD 75 02   CMP $0275        Characters to be searched for
C276   F0 28      BEQ $C2A0        Identical w/chars. in input line?
C278   C9 2A      CMP #$2A         NO—Compare with wildcard ('*')
C27A   F0 04      BEQ $C280        Identical?
C27C   C9 3F      CMP #$3F         NO—Compare with wildcard ('?')
C27E   D0 03      BNE $C283        Identical?
C2801  EE 8A 02   INC $028A        YES—Set wildcard flag
C2831  C9 2C      CMP #$2C         Compare current char with ','
C285   D0 E4      BNE $C26B        Identical?
C287   98         TYA              YES—Save comma position+1 as
C288   9D 7B 02   STA $027B,X      start- position of next parameter
C28B   AD 8A 02   LDA $028A        Get wildcard flag back
C28E   29 7F      AND #$7F         Clear wildcard flag
C290   F0 07      BEQ $C299        Find a joker?
C292   A9 80      LDA #$80         YES—Set bit 7 as flag and
C294   95 E7      STA $E7,X        save filename as name with joker
C296   8D 8A 02   STA $028A        Set bit 7 of wildcard flag
C2991  E8         INX              Increment # of parameters found,
C29A   E0 04      CPX #$04         separated by commas; reached a
C29C   90 CD      BCC $C26B        maximum of five open files?
C29E1  A0 00      LDY #$00         YES—Y-value=0 signalling end
C2A01  AD 74 02   LDA $0274        Save length of command line as
C2A3   9D 7B 02   STA $027B,X      start position of last parameter
C2A6   AD 8A 02   LDA $028A        Get wildcard flg of last filename
C2A9   29 7F      AND #$7F         Remove bit 7
C2AB   F0 04      BEQ $C2B1        Wildcard in parameter onhand?
C2AD   A9 80      LDA #$80         YES—Identify in file table as
C2AF   95 E7      STA $E7,X        name with a wildcard attached
2B11   98         TYA              Position in input line (end=0)
C2B2   60         RTS              Return from this subroutine
-----------------------------------------------------------------------
```

```
[BF48/C160/D7B9/E207/C2DC:BF4B,DA86]
Set all flags and look at command string table
C2B3    A4 A3       LDY $A3         Low-byte of input buffer pointer
C2B5    F0 14       BEQ $C2CB       address $A3/$A4 =$0200?
C2B7    88          DEY             NO—Adjust buffer pointer
C2B8    F0 10       BEQ $C2CA       Is $A3 =1?
C2BA    20 02 AA    JSR $AA02       NO—Test for User command
C2BD    C9 0D       CMP #$0D        Check command chars for <RETURN>
C2BF    F0 0A       BEQ $C2CB       End-of-line reached?
C2C1    88          DEY             NO—Set pointer to character
C2C2    B9 00 02    LDA $0200,Y     Get char from buffer; compare
C2C5    C9 0D       CMP #$0D        with <RETURN> (end-of-line)
C2C7    F0 02       BEQ $C2CB       Identical?
C2C9    C8          INY             NO—Set buffer pointer back to
C2CA[1] C8          INY             output value
C2CB[3] 8C 74 02    STY $0274       Save pointer value to end of cmd
C2CE    C0 2A       CPY #$2A        string; max. length reached?
C2D0    A0 FF       LDY #$FF        Value of 'no command' command #
C2D2    90 08       BCC $C2DC       Cmd string less than buffer(42)?
C2D4    8C 2A 02    STY $022A       NO—Clear command number
C2D7    A9 32       LDA #$32        Line too long?
C2D9    4C C8 C1    JMP $C1C8       Display '32 Syntax Error' message
C2DC[3] A0 00       LDY #$00        Clear & set back table, pointers
C2DE    98          TYA             and flags
C2DF    85 A3       STA $A3         Input buffer pointer at $0200
C2E1    8D 58 02    STA $0258       Current record length
C2E4    8D 4A 02    STA $024A       Current filetype
C2E7    8D 96 02    STA $0296       Filetype from command string
C2EA    85 D3       STA $D3         Pointer to first filename
C2EC    8D 79 02    STA $0279       Filename pointer
C2EF    8D 77 02    STA $0277       #of files for 1stfile designation
C2F2    8D 78 02    STA $0278       #of files for 2ndfile designation
C2F5    8D 8A 02    STA $028A       'Wildcard found in filename' flag
C2F8    8D 6C 02    STA $026C       'Syntax Error' flag
C2FB    A2 05       LDX #$05        Clear table to five filenames
C2FD[1] 9D 79 02    STA $0279,X     End position of filename in buffr
C300    95 D7       STA $D7,X       Directory sector of file
C302    95 DC       STA $DC,X       Position/file in directory sector
C304    95 E1       STA $E1,X       Drive number of file
C306    95 E6       STA $E6,X       Filetype and wildcard flag
C308    9D 7F 02    STA $027F,X     Current track number of file
C30B    9D 84 02    STA $0284,X     Current sector number of file
C30E    CA          DEX             Table for next filename
C30F    D0 EC       BNE $C2FD       All 5 possible filenames ready?
C311    60          RTS             Return from this subroutine
-------------------------------------------------------------------
```

```
[90B8/A60E/D84C/EE0D/C320:C826,C90F,CA88,DA96]
Get drive number of file and set into file table
C312  AD 78 02   LDA $0278      Recover number of files for
C315  8D 77 02   STA $0277      file specification
C318  A9 01      LDA #$01       Set number to be given for
C31A  8D 78 02   STA $0278      a file
C31D  8D 79 02   STA $0279      indication
C320⁴ AC 8E 02   LDY $028E      Lzast active drive
C323  A2 00      LDX #$00       Save current number
C325⁵ 86 D3      STX $D3        of filename
C327  BD 7A 02   LDA $027A,X    Starting pos.of filename in buffr
C32A  20 3C C3   JSR $C33C      Get drive number from buffer
C32D  A6 D3      LDX $D3        Number of current file parameters
C32F  9D 7A 02   STA $027A,X    Save position in command string;
C332  98         TYA            Get drive position from  that
C333  95 E2      STA $E2,X      Set drive number for file
C335  E8         INX            Go to next file
C336  EC 78 02   CPX $0278      Compare with # of files to be
C339  90 EA      BCC $C325      worked on; all of them ready?
C33B  60         RTS            YES—Return from this subroutine
--------------------------------------------------------------------
[C32A]
Get drive number from command string
C33C  AA         TAX            Position of filename in buffer
C33D  A0 00      LDY #$00       Number of standard drive
C33F  A9 3A      LDA #$3A       Colon ':'
C341  DD 01 02   CMP $0201,X    Colon behind current position?
C344  F0 0C      BEQ $C352      YES—Then syntax correct/goto$C352
C346  DD 00 02   CMP $0200,X    Compare with current position
C349  D0 01      BNE $C361      Pointer aiming at a colon?
C34B  E8         INX            YES—No drive assignment
C34C¹ 98         TYA            Drive number
C34D² 29 01      AND #$01       (0 or 1 allowed only);
C34F¹ A8         TAY            save drive in Y-register
C350  8A         TXA            Current position in input buffer
C351  60         RTS            Return from this subroutine
C352¹ BD 00 02   LDA $0200,X    Drive number from command string
C355  E8         INX            Set input buffer pointer behind
C356  E8         INX            drives indicator (':')
C357  C9 30      CMP #$30       Drive null?
C359  F0 F2      BEQ $C34D      YES—Then set drive;
C35B  C9 31      CMP #$31       Test for drive 1
C35D  F0 EE      BEQ $C34D      Identical?
```

```
C35F   D0 EB      BNE $C34C       NO—Jump to $C34C
C361¹  98         TYA             Standard drive number (0)
C362   09 80      ORA #$80        Set flag for improper
C364   29 81      AND #$81        drive number and give to
C366   D0 E7      BNE $C34F       subroutine
-----------------------------------------------------------
[C1E2]
Initialize drive and switch drive LED on
C368   A9 00      LDA #$00        Clear command syntax
C36A   8D 8B 02   STA $028B       flag
C36D   AC 7A 02   LDY $027A       Current position in input buffer
C370¹  B1 A3      LDA ($A3),Y     Get character from command string
C372   20 BD C3   JSR $C3BD       Test for legal drive number
C375   10 11      BPL $C388       Number correct?
C377   C8         INY             NO—Pointer to next character
C378   CC 74 02   CPY $0274       Length of command string
C37B   B0 06      BCS $C383       Reached end?
C37D   AC 74 02   LDY $0274       NO—Length of command string
C380   88         DEY             Set pointer to last character
C381   D0 ED      BNE $C370       Is only 1 command char onhand?
C383¹  CE 8B 02   DEC $028B       YES—'not found' in syntax flag
C386   A9 00      LDA #$00        Set number of standard drive
C388¹  29 01      AND #$01        as current drive number
C38A   85 7F      STA $7F         --
C38C   4C 00 C1   JMP $C100       LED on
-----------------------------------------------------------
[C40E/C420/C427/C467/C497/C704/C70B]
Switch to other drive
C38F   A5 7F      LDA $7F         Current drive number
C391   49 01      EOR #$01        Turn to drive bit and remove
C393   29 01      AND #$01        other bits
C395   85 7F      STA $7F         Store as current drive
C397   60         RTS             Return from this subroutine
-----------------------------------------------------------
[C823/DA98]
Set and determine filetype
C398   A0 00      LDY #$00        Choose first filename for table
C39A   AD 77 02   LDA $0277       Check position of 1st filename
C39D   CD 78 02   CMP $0278       w/position of filetype identifier
C3A0   F0 16      BEQ $C3B8       Identical?
C3A2   CE 78 02   DEC $0278       NO—Then set pointer to filetype
C3A5   AC 78 02   LDY $0278       --
C3A8   B9 7A 02   LDA $027A,Y     Take pointer to end of filename
C3AB   A8         TAY             and get matching characters
C3AC   B1 A3      LDA ($A3),Y     from the filename
C3AE   A0 04      LDY #$04        Number of possible filetypes
```

```
C3B0¹ D9 BB FE    CMP $FEBB,Y      Characters in filetype table
C3B3  F0 03       BEQ $C3B8        onhand?
C3B5  88          DEY              Turn pointer to next filetype
C3B6  D0 F8       BNE $C3B0        Already testd with all filetypes?
C3B8² 98          TYA              YES—Save number of
C3B9  8D 96 02    STA $0296        filetypes (=0 when none exist)
C3BC  60          RTS              Return from this subroutine
----------------------------------------------------------------
[C372/DA68]
Drive number tested for validity
C3BD  C9 30       CMP #$30         Is drive number
C3BF  F0 06       BEQ $C3C7        equal to drive 0?
C3C1  C9 31       CMP #$31         NO—Then is drive number
C3C3  F0 02       BEQ $C3C7        equal to drive 1?
C3C5  09 80       ORA #$80         NO—Set bit 7 as error flag &clear
C3C7² 29 81       AND #$81         remaining bits(from ASCII values)
C3C9  60          RTS              Return from this subroutine
----------------------------------------------------------------
[C44F/C829/D84F/DA9E]
Initialize drive given in filename
C3CA  A9 00       LDA #$00         Clear temp. memory for creating
C3CC  85 6F       STA $6F          index of control bytes
C3CE  8D 8D 02    STA $028D        Clear 'number of drives' flag
C3D1  48          PHA              Prepare stack for following prog.
C3D2  AE 78 02    LDX $0278        Number of names going with files
C3D5² 68          PLA              Pointer to control byte
C3D6  05 6F       ORA $6F          Enter and save last
C3D8  48          PHA              entry
C3D9  A9 01       LDA #$01         Set flag for 'Drive indication
C3DB  85 6F       STA $6F          is onhand'
C3DD  CA          DEX              Countr for filenames tobe checked
C3DE  30 0F       BMI $C3EF        Dirve number ready for all files?
C3E0  B5 E2       LDA $E2,X        Get file drive number
C3E2  10 04       BPL $C3E8        Drives identifier set?
C3E4  06 6F       ASL $6F          NO—Adjust
C3E6  06 6F       ASL $6F          bit flags
C3E8¹ 4A          LSR A            Test drive number
C3E9  90 EA       BCC $C3D5        Is drive 1 chosen?
C3EB  06 6F       ASL $6F          YES—Pointer to bytes for drive 1
C3ED  D0 E6       BNE $C3D5        Jump to $C3D5
C3EF¹ 68          PLA              Set control byte pointer
C3F0  AA          TAX              for drive initialization
C3F1  BD 3F C4    LDA $C43F,X      Get and save access
C3F4  48          PHA              control byte
C3F5  29 03       AND #$03         Determine # of allowable drives;
C3F7  8D 8C 02    STA $028C        save it
```

```
C3FA   68          PLA         Repeat control byte
C3FB   0A          ASL A       Flag for 'Only one drive'
C3FC   10 3E       BPL $C43C   Is only one indicator allowed?
C3FE   A5 E2       LDA $E2     YES—Take drive # of first file
C400   29 01       AND #$01    as current
C402   85 7F       STA $7F     drive
C404   AD 8C 02    LDA $028C   Get number of allowable drives
C407   F0 2B       BEQ $C434   Is it an allowable drive?
C409   20 3D C6    JSR $C63D   NO—Initialize current drive
C40C   F0 12       BEQ $C420   Drive ready?
C40E   20 8F C3    JSR $C38F   NO—Switch to other drive
C411   A9 00       LDA #$00    Clear number of allowable
C413   8D 8C 02    STA $028C   drives
C416   20 3D C6    JSR $C63D   Initialize other drive
C419   F0 1E       BEQ $C439   Drive ready?
C41B1  A9 74       LDA #$74    NO—Display
C41D   20 C8 C1    JSR $C1C8   '74 Drive Not Ready' message
C4201  20 8F C3    JSR $C38F   Change to other drive
C423   20 3D C6    JSR $C63D   Initialize drive and
C426   08          PHP         save result
C427   20 8F C3    JSR $C38F   Switch to other drive & get
C42A   28          PLP         previous result again
C42B   F0 0C       BEQ $C439   Is previous drive ready?
C42D   A9 00       LDA #$00    NO—CLear legal number of
C42F   8D 8C 02    STA $028C   drives
C432   F0 05       BEQ $C439   Jump to $C439
C4341  20 3D C6    JSR $C63D   Initialize drive
C437   D0 E2       BNE $C41B   Is drive ready?
C4393  4C 00 C1    JMP $C100   YES—Switch LED of drive on
C43C1  2A          ROL A       Adjust control byte and get
C43D   4C 00 C4    JMP $C400   drive number from control byte
------------------------------------------------------------
[C3F1]  Control bytes for type of drive initialization

Functions of individual bits:
Bit 0/1 : Number of drives utilized (0/1/2)
Bit 6   : 1= Take drive number from control byte
Bit 7   : 0/1 drive number for Bit6

C440   00 80 41 01 01 01 01 81
C448   81 81 81 42 42 42 42
------------------------------------------------------------
```

```
[90C8/C952/CA99/E7B8]
Look for file entry in directory
C44F    20 CA C3   JSR $C3CA    Set disk drive for file to search
C452[1] A9 00      LDA #$00     Indicator on 1st directory entry
C454    8D 92 02   STA $0292    Erase
C457    20 AC C5   JSR $C5AC    Set indicator, search entry
C45A    D0 19      BNE $C475    Valid entry found?
C45C[1] CE 8C 02   DEC $028C    NO—number of disk drives
C45F    10 01      BPL $C462    One more disk drive?
C461    60         RTS          Back to calling routine
C462[1] A9 01      LDA #$01     Flag for both disk drives
C464    8D 8D 02   STA $028D    Search set
C467    20 8F C3   JSR $C38F    Change to other disk drive
C46A    20 00 C1   JSR $C100    Activate LED at disk drive
C46D    4C 52 C4   JMP $C452    Search entry on other disk drive
C470[1] 20 17 C6   JSR $C617    Search next valid file
C473    F0 10      BEQ $C485    Found?
C475[2] 20 D8 C4   JSR $C4D8    YES—check directory entry
C478    AD 8F 02   LDA $028F    Indicator for file entry found
C47B    F0 01      BEQ $C47E    Is entry corrrect?
C47D    60         RTS          YES—back to calling routine
C47E[1] AD 53 02   LDA $0253    Flag for entry is found
C481    30 ED      BMI $C470    Is file found?
C483    10 F0      BPL $C475    YES— jump to $C475
C485[1] AD 8F 02   LDA $028F    Is flag for file found
C488    F0 D2      BEQ $C45C    Last entry?
C48A    60         RTS          NO—return to the calling routine
----------------------------------------------------------------------
[C86D]
Search file entry in directory
C48B    20 04 C6   JSR $C604    Search directory for file
C48E    F0 1A      BEQ $C4AA    Is entry found?
C490    D0 28      BNE $C4BA    YES—continue at $C4BA
C492[1] A9 01      LDA #$01     Flag for access on both drives
C494    8D 8D 02   STA $028D    Set
C497    20 8F C3   JSR $C38F    Change to other disk drive
C49A    20 00 C1   JSR $C100    Switch on LED
C49D[3] A9 00      LDA #$00     Indicator on first value
C49F    8D 92 02   STA $0292    Delete entry
C4A2    20 AC C5   JSR $C5AC    Initial indicator; search entry
C4A5    D0 13      BNE $C4BA    Is file found?
C4A7    8D 8F 02   STA $028F    Position
C4AA[2] AD 8F 02   LDA $028F    Store
C4AD    D0 28      BNE $C4D7    Last entry
C4AF    CE 8C 02   DEC $028C    YES—number of allowed drives
C4B2    10 DE      BPL $C492    Switch to other disk drive?
```

```
C4B4   60          RTS                 NO—return to calling routine
C4B5³  20 17 C6    JSR $C617           Get next entry
C4B8   F0 F0       BEQ $C4AA           Entry found?
C4BA²  20 D8 C4    JSR $C4D8           YES—verify entry w/searched flag
C4BD   AE 53 02    LDX $0253           Get flag
C4C0   10 07       BPL $C4C9           Is entry the same?
C4C2   AD 8F 02    LDA $028F           NO—get flag found for file
C4C5   F0 EE       BEQ $C4B5           Was file found?
C4C7   D0 0E       BNE $C4D7           NO—jump to $C4D7
C4C9¹  AD 96 02    LDA $0296           Get actual file type
C4CC   F0 09       BEQ $C4D7           Is entry valid?
C4CE   B5 E7       LDA $E7,X           YES—get indicator:search filetype
C4D0   29 07       AND #$07            And insulate save for type
C4D2   CD 96 02    CMP $0296           Verify with search file type
C4D5   D0 DE       BNE $C4B5           Identical?
C4D7³  60          RTS                 YES—return to calling routine
------------------------------------------------------------------
[C475/C4BA]
Search directory entry
C4D8   A2 FF       LDX #$FF            Flag for entry found
C4DA   8E 53 02    STX $0253           Delete
C4DD   E8          INX                 (0)
C4DE   8E 8A 02    STX $028A           Delete flag for wildcard
C4E1   20 89 C5    JSR $C589           Set file flag
C4E4   F0 06       BEQ $C4EC           Was entry found?
C4E6¹  60          RTS                 YES—return to calling routine
------------------------------------------------------------------
[C4F5/C4FC/C513/C519/C533]
Search next entry
C4E7   20 94 C5    JSR $C594           Search next entry
C4EA   D0 FA       BNE $C4E6           Found ?
C4EC¹  A5 7F       LDA $7F             YES—get current disk drive and
C4EE   55 E2       EOR $E2,X           Verify with disk drive number of
C4F0   4A          LSR A               the file entry
C4F1   90 0B       BCC $C4FE           Identical?
C4F3   29 40       AND #$40            NO—get flag for disk drive type
C4F5   F0 F0       BEQ $C4E7           Drive set with standard value?
C4F7   A9 02       LDA #$02            YES—value f/access to both drives
C4F9   CD 8C 02    CMP $028C           Verify with access flag
C4FC   F0 E9       BEQ $C4E7           Search for both disk?
C4FE¹  BD 7A 02    LDA $027A,X         NO—get and store position of file
C501   AA          TAX                 Name in command string
C502   20 A6 C6    JSR $C6A6           Set parameter for name in command
C505   A0 03       LDY #$03            Set buffer indicator on Dir. name
C507   4C 1D C5    JMP $C51D           Verify names with command string
C50A¹  BD 00 02    LDA $0200,X         Get character from oommand string
```

```
C50D  D1 94     CMP ($94),Y   Verify name with directory name
C50F  F0 0A     BEQ $C51B     Identical?
C511  C9 3F     CMP #$3F      NO—verify wilcard with '?'
C513  D0 D2     BNE $C4E7     Identical?
C515  B1 94     LDA ($94),Y   YES—get char from directory entry
C517  C9 A0     CMP #$A0      Verify w/value for shift space
C519  F0 CC     BEQ $C4E7     Entire  filename already read?
C51B1 E8        INX           NO—indicator of command string
C51C  C8        INY           Point idicator in directory buff
C51D1 EC 76 02  CPX $0276     Indicator to end of name in comnd
C520  B0 09     BCS $C52B     End of filename reached?
C522  BD 00 02  LDA $0200,X   NO—get character from filename
C525  C9 2A     CMP #$2A      Verify with '*'char for wildcard
C527  F0 0C     BEQ $C535     Identical?
C529  D0 DF     BNE $C50A     NO—jump to $C50A
C52B1 C0 13     CPY #$13      Verify with return
C52D  B0 06     BCS $C535     Is  ASCII value smaller ?
C52F  B1 94     LDA ($94),Y   YES—get char from directory and
C531  C9 A0     CMP #$A0      Verify with value for shift space
C533  D0 B2     BNE $C4E7     Complete filename already read?
C5352 AE 79 02  LDX $0279     YES—get position for dir. entry
C538  8E 53 02  STX $0253     and set indicator
C53B  B5 E7     LDA $E7,X     Determine entry of file
C53D  29 80     AND #$80      Set flag for joker
C53F  8D 8A 02  STA $028A     and store
C542  AD 94 02  LDA $0294     Indicator to pos. in dir. buffer
C545  95 DD     STA $DD,X     Determine filename in table
C547  A5 81     LDA $81       Number of directory sector
C549  95 D8     STA $D8,X     Store
C54B  A0 00     LDY #$00      Buffer indicator to entry start
C54D  B1 94     LDA ($94),Y   Get file type from directory
C54F  C8        INY           Buffer indicator to next char
C550  48        PHA           Store original file type
C551  29 40     AND #$40      Insulate flag for scratch-protect
C553  85 6F     STA $6F       and store
C555  68        PLA           Recall file type
C556  29 DF     AND #$DF      Fade out scratch file
C558  30 02     BMI $C55C     Is file closed properly?
C55A  09 20     ORA #$20      NO—flag for'*' file
C55C1 29 27     AND #$27      Take over flag and file type
C55E  05 6F     ORA $6F       and fade in scratch flag
C560  85 6F     STA $6F       Store both
C562  A9 80     LDA #$80      Flag for 'File type is set'
C564  35 E7     AND $E7,X     Take over from table
C566  05 6F     ORA $6F       & fade in bits from new filetypes
C568  95 E7     STA $E7,X     Determine type in table/filename
```

```
C56A  B5 E2     LDA $E2,X       Get number of entry's disk drive
C56C  29 80     AND #$80        and current disk drive number
C56E  05 7F     ORA $7F         Enter
C570  95 E2     STA $E2,X       Write value in disk drive table
C572  B1 94     LDA ($94),Y     Get track number of 1st sector
C574  9D 80 02  STA $0280,X     and enter in table
C577  C8        INY             Point indicator to next byte
C578  B1 94     LDA ($94),Y     Get sector number from entry
C57A  9D 85 02  STA $0285,X     and store
C57D  AD 58 02  LDA $0258       Get current record length
C580  D0 07     BNE $C589       Is value set?
C582  A0 15     LDY #$15        NO—buffr indicator to value/entry
C584  B1 94     LDA ($94),Y     Get record length of dir. entry
C586  8D 58 02  STA $0258       and store in indicator
-----------------------------------------------------------------
[C4E1/C580]
Re-initial flags
C589  A9 FF     LDA #$FF        Delete indicator
C58B  8D 8F 02  STA $028F       Flag for last entry
C58E  AD 78 02  LDA $0278       Indicator to position of filename
C591  8D 79 02  STA $0279       in the input buffer
-----------------------------------------------------------------
[C4E7/C5A4]
Quit search for filename
C594  CE 79 02  DEC $0279       Number of filenames
C597  10 01     BPL $C59A       Process more entries?
C599  60        RTS             NO—back to calling routine
C59A1 AE 79 02  LDX $0279       Get number of filename
C59D  B5 E7     LDA $E7,X       and determine current file type
C59F  30 05     BMI $C5A6       Is value set?
C5A1  BD 80 02  LDA $0280,X     NO—get track of first sector
C5A4  D0 EE     BNE $C594       Is value determined?
C5A61 A9 00     LDA #$00        NO—flag:last dir entry reached
C5A8  8D 8F 02  STA $028F       Set
C5AB  60        RTS             Return to calling routine
-----------------------------------------------------------------
[C457/C4A2/D70E/ED97]
Set indicator to search in directory
C5AC  A0 00     LDY #$00        Indicator to current dir. sector
C5AE  8C 91 02  STY $0291       Delete
C5B1  88        DEY             Flag for 'ENTRY FOUND'
C5B2  8C 53 02  STY $0253       Reset
C5B5  AD 85 FE  LDA $FE85       Number of directory track(18)
C5B8  85 80     STA $80         Get, and store as current track
C5BA  A9 01     LDA #$01        Indicator to sector number
C5BC  85 81     STA $81         Set
```

```
C5BE    8D 93 02   STA $0293      Delete flag for 'SECTOR READ'
C5C1    20 75 D4   JSR $D475      Transfer sector in buffer
C5C4[1] AD 93 02   LDA $0293      Indicator to next dir. sector
C5C7    D0 01      BNE $C5CA      Is there another sector?
C5C9    60         RTS            NO—return to calling routine
C5CA[1] A9 07      LDA #$07       Store amount of file entries in
C5CC    8D 95 02   STA $0295      a directory block - 1
C5CF    A9 00      LDA #$00       Position of byte to read
C5D1    20 F6 D4   JSR $D4F6      Get byte of current buffer
C5D4    8D 93 02   STA $0293      and store
C5D7[1] 20 E8 D4   JSR $D4E8      Set indicator for current buffer
C5DA    CE 95 02   DEC $0295      Counter for entries in dir.sector
C5DD    A0 00      LDY #$00       Set indicator to start of entry
C5DF    B1 94      LDA ($94),Y    and get file type identification
C5E1    D0 18      BNE $C5FB      Is entry deleted?
C5E3    AD 91 02   LDA $0291      YES—number of current sector
C5E6    D0 2F      BNE $C617      Is value set?
C5E8    20 3B DE   JSR $DE3B      NO—get track and sector number
C5EB[1] A5 81      LDA $81        # of current directory sector
C5ED    8D 91 02   STA $0291      Store
C5F0    A5 94      LDA $94        Low-byte of indicator to entry
C5F2    AE 92 02   LDX $0292      Get indicator to valid entry
C5F5    8D 92 02   STA $0292      Set new value
C5F8    F0 1D      BEQ $C617      Was indicator deleted before?
C5FA    60         RTS            NO—return to calling routine
C5FB*   A2 01      LDX #$01       Number of first entry
C5FD    EC 92 02   CPX $0292      Verify with last value
C600    D0 2D      BNE $C62F      Was first entry set?
C602    F0 13      BEQ $C617      NO—jump to $0617
C604[2] AD 85 FE   LDA $FE85      Number of directory track
C607    85 80      STA $80        Get and store as current track
C609    AD 90 02   LDA $0290      Number of directory sector
C60C    85 81      STA $81        Store as current sector
C60E    20 75 D4   JSR $D475      Read sector from disc in buffer
C611    AD 94 02   LDA $0294      Indictor on postion of entry
C614    20 C8 D4   JSR $D4C8      Set directory indicator
C617[6] A9 FF      LDA #$FF       Flag for 'file entry found'
C619    8D 53 02   STA $0253      Delete
C61C    AD 95 02   LDA $0295      # of directory entries per sector
C61F    30 08      BMI $C629      Is counter set?
C621    A9 20      LDA #$20       YES—amount of bytes of an entry
C623    20 C6 D1   JSR $D1C6      Buff indictor to next file entry
C626    4C D7 C5   JMP $C5D7      Set indicator
C629[1] 20 4D D4   JSR $D44D      Read next block from directory
C62C    4C C4 C5   JMP $C5C4      Set indicator
```

```
C62F1 A5 94     LDA $94       Low-byte of current indictor
C631  8D 94 02  STA $0294     Store
C634  20 3B DE  JSR $DE3B     Get track and sector of last job
C637  A5 81     LDA $81       Number of directory sector
C639  8D 90 02  STA $0290     Store
C63C  60        RTS           Back to calling routine
-----------------------------------------------------------------
[83A0/840B/9088/C409/C416/C423/C434/CB8C]
Initial Diskette
C63D  A5 68     LDA $68       Flag for 'initial automatic'
C63F  D0 28     BNE $C669     Initial permitted only by hand?
C641  A6 7F     LDX $7F       NO—number of current disc drive
C643  56 1C     LSR $1C,X     Get/check flag for initialization
C645  90 22     BCC $C669     Shall disc be initialized?
C647  A9 FF     LDA #$FF      Yes,watch flag for 'error by job'
C649  8D 98 02  STA $0298     Delete
C64C  20 0E D0  JSR $D00E     Check if disk is inserted
C64F  A0 FF     LDY #$FF      Flag value for 'error occured'
C651  C9 02     CMP #$02      Verify result with code for sync
C653  F0 0A     BEQ $C65F     Was sync mark found?
C655  C9 03     CMP #$03      YES—check for blockheader code
C657  F0 06     BEQ $C65F     Was blockheader found?
C659  C9 0F     CMP #$0F      YES—check code of disk drive
C65B  F0 02     BEQ $C65F     Is the disk drive approachable?
C65D  A0 00     LDY #$00      YES—flag value for "no error'
C65F3 A6 7F     LDX $7F       Get number of current disk drive
C661  98        TYA           And the error flag
C662  95 FF     STA $FF,X     In respective disk drive status
C664  D0 03     BNE $C669     Is disk drive ready?
C666  20 42 D0  JSR $D042     YES—read BAM
C6693 A6 7F     LDX $7F       Get number of current disc drive
C66B  B5 FF     LDA $FF,X     And respective disc drive status
C66D  60        RTS           Back to calling routine
-----------------------------------------------------------------
[CAC0/D768/EE68]
Copy file name from input buffer to directory buffer
(The Accumulator has to contain the length of the name in X, the
position in the string in Y and the number of the directory buffer)
C66E  48        PHA           Store length of file name
C66F  20 A6 C6  JSR $C6A6     Postition of name in inputstring
C672  20 88 C6  JSR $C688     Determine and copy name in buffer
C675  68        PLA           Recall length of file name
C676  38        SEC           Length of copied file name
C677  ED 4B 02  SBC $024B     Verify with maximum lengths
C67A  AA        TAX           of file entry (16)
C67B  F0 0A     BEQ $C687     Is entry fulfilled?
C67D  90 08     BCC $C687     No—file name smaller than place?
```

```
C67F   A9 A0      LDA #$A0       Yes—fill rest of the characters
C681¹  91 94      STA ($94),Y    of filename with 'shift space'
C683   C8         INY            Buffer indicator to nxt char pos.
C684   CA         DEX            Amount of characters to fill
C685   D0 FA      BNE $C681      File named filled?
C687²  60         RTS            YES—back to calling routine
----------------------------------------------------------------
[C672] Copy part of the input buffer and the current data buffer
C688   98         TYA            Double the number
C689   0A         ASL A          of current buffer
C68A   A8         TAY            (Table contains 2-byte indicator
C68B   B9 99 00   LDA $0099,Y    Get address of buffer (Low-byte)
C68E   85 94      STA $94        & set in indicator of dir. buffer
C690   B9 9A 00   LDA $009A,Y    Get address of buffer (High-byte)
C693   85 95      STA $95        & set in indicator in dir. buffer
C695   A0 00      LDY #$00       Delete indicatr on bufferposition
C697¹  BD 00 02   LDA $0200,X    Get byte from input buffer
C69A   91 94      STA ($94),Y    And copy in current buffer
C69C   C8         INY            Data buffr indicator to next page
C69D   F0 06      BEQ $C6A5      Data buffer full?
C69F   E8         INX            NO—raise indicator input buffer
C6A0   EC 76 02   CPX $0276      Verify w/lengths of comnd strings
C6A3   90 F2      BCC $C697      Last character reached?
C6A5¹  60         RTS            YES—back to calling routine
----------------------------------------------------------------
[C502/C66F]
Search length of file name in input buffer (Start position in X)
C6A6   A9 00      LDA #$00       Indicator on length of name
C6A8   8D 4B 02   STA $024B      Delete
C6AB   8A         TXA            Get startposition in input buffer
C6AC   48         PHA            And store
C6AD¹  BD 00 02   LDA $0200,X    Get character of name
C6B0   C9 2C      CMP #$2C       And verify with ','
C6B2   F0 14      BEQ $C6C8      Indentical?
C6B4   C9 3D      CMP #$3D       NO—verify with '='
C6B6   F0 10      BEQ $C6C8      Identical?
C6B8   EE 4B 02   INC $024B      NO—raise length of file name
C6BB   E8         INX            Set buffer indicator to next char
C6BC   A9 0F      LDA #$0F       Verify lengths of current name
C6BE   CD 4B 02   CMP $024B      with maximum lengths of file name
C6C1   90 05      BCC $C6C8      Current file name to big?
C6C3   EC 74 02   CPX $0274      NO—verify position w/endof string
C6C6   90 E5      BCC $C6AD      Is end of input string reached?
C6C8³  8E 76 02   STX $0276      YES—store length of file name
C6CB   68         PLA            & recall indicator to start pos.
C6CC   AA         TAX            And set
C6CD   60         RTS            Back to calling routine
```

```
[ECF2]
Read file entry from directory
C6CE  A5 83     LDA $83        Current active secondary address
C6D0  48        PHA            Save
C6D1  A5 82     LDA $82        Number of current active channel
C6D3  48        PHA            Save
C6D4  20 DE C6  JSR $C6DE      Get file entry
C6D7  68        PLA            Number of channel
C6D8  85 82     STA $82        Restore
C6DA  68        PLA            End previous secondary address
C6DB  85 83     STA $83        Set again
C6DD  60        RTS            Back to calling routine
-------------------------------------------------------------
[C6D4]
Establish directory for output to buffer
C6DE  A9 11     LDA #$11       Secondary address 17
C6E0  85 83     STA $83        Determine
C6E2  20 EB D0  JSR $D0EB      Open channel
C6E5  20 E8 D4  JSR $D4E8      Set indicator to current buffer
C6E8  AD 53 02  LDA $0253      Get flag for file entry
C6EB  10 0A     BPL $C6F7      Was entry found?
C6ED  AD 8D 02  LDA $028D      NO—flag for directory disc drives
C6F0  D0 0A     BNE $C6FC      Directory of both disc drives?
C6F2  20 06 C8  JSR $C806      NO—get 'blocks free' and write
C6F5  18        CLC            In buffer
C6F6  60        RTS            Back to calling routine
C6F7¹ AD 8D 02  LDA $028D      Check flag for directory drives
C6FA  F0 1F     BEQ $C71B      Directory of both disc drives
C6FC¹ CE 8D 02  DEC $028D      YES—set flag to 'no'
C6FF  D0 0D     BNE $C70E      Was flag not set right?
C701  CE 8D 02  DEC $028D      NO—correct flag
C704  20 8F C3  JSR $C38F      Change to other disc drive
C707  20 06 C8  JSR $C806      Get 'blocks free' write in buffer
C70A  38        SEC            And switch back to
C70B  4C 8F C3  JMP $C38F      Previous disk drive
C70E¹ A9 00     LDA #$00       Memory for block number
C710  8D 73 02  STA $0273      Delete
C713  8D 8D 02  STA $028D      Delete flag: 'both disc drives'
C716  20 B7 C7  JSR $C7B7      Develope title of directory
C719  38        SEC            Flag for 'more entries'
C71A  60        RTS            Back to calling routine
C71B¹ A2 18     LDX #$18       Length of directory line (24)
C71D  A0 1D     LDY #$1D       Set byte indicator for filelength
C71F  B1 94     LDA ($94),Y    Get amount of blocks (high-byte)
C721  8D 73 02  STA $0273      and store
C724  F0 02     BEQ $C728      Is block number >256 & 3 digit?
C726  A2 16     LDX #$16       YES—decrease length of characters
```

```
C728[1] 88          DEY              Buffer indicator for block #
C729    B1 94       LDA ($94),Y      Get lo-byte for block number
C72B    8D 72 02    STA $0272        And store
C72E    E0 16       CPX #$16         Verify w/value for short length
C730    F0 0A       BEQ $C73C        Is 3 digit block number there?
C732    C9 0A       CMP #$0A         Verify amount of blocks with ten
C734    90 06       BCC $C73C        Block number smaller (one digit)?
C736    CA          DEX              NO—shorten rest line
C737    C9 64       CMP #$64         Verify block number with 100
C739    90 01       BCC $C73C        Is block number smaller(2 digit)?
C73B    CA          DEX              NO—shorten rest line
C73C[3] 20 AC C7    JSR $C7AC        Delete buffer for directory
C73F    B1 94       LDA ($94),Y      Get byte for file type
C741    48          PHA              And store
C742    0A          ASL A            Get bit 6 as flag file lock
C743    10 05       BPL $C74A        Is file locked?
C745    A9 3C       LDA #$3C         YES—char for file locking '<'
C747    9D B2 02    STA $02B2,X      Write behind file type
C74A[1] 68          PLA              Recall file type
C74B    29 0F       AND #$0F         Insulate file type
C74D    A8          TAY              And its short name in directory
C74E    B9 C5 FE    LDA $FEC5,Y      3rd letter of file short name
C751    9D B1 02    STA $02B1,X      Get and write in buffer
C754    CA          DEX              Shorten length of directory line
C755    B9 C0 FE    LDA $FEC0,Y      2nd letter of file short name
C758    9D B1 02    STA $02B1,X      Get and write in buffer
C75B    CA          DEX              Shorten name of directory line
C75C    B9 BB FE    LDA $FEBB,Y      1st letter of file short name
C75F    9D B1 02    STA $02B1,X      Get and write in buffer
C762    CA          DEX              Shorten length of
C763    CA          DEX              Directory
C764    B0 05       BCS $C76B        Is the file closed properly?
C766    A9 2A       LDA #$2A         NO—'*' as notification
C768    9D B2 02    STA $02B2,X      Set before file short name
C76B[1] A9 A0       LDA #$A0         One empty character
C76D    9D B1 02    STA $02B1,X      Insert
C770    CA          DEX              & shorten length of dir. line
C771    A0 12       LDY #$12         Set buffer position of file name
C773[1] B1 94       LDA ($94),Y      Get character of file name
C775    9D B1 02    STA $02B1,X      And write in directory buffer
C778    CA          DEX              Shorten length of directory line
C779    88          DEY              Lower buffer indicator
C77A    C0 03       CPY #$03         Verify with end value
C77C    B0 F5       BCS $C773        All chars of name taken over?
C77E    A9 22       LDA #$22         YES—set "before name"
C780    9D B1 02    STA $02B1,X      Set
C783[1] E8          INX              Raise indicator in directory line
```

```
C784   E0 20      CPX #$20       Check to maximum value
C786   B0 0B      BCS $C793      End of buffer reached?
C788   BD B1 02   LDA $02B1,X    NO—get character from file name
C78B   C9 22      CMP #$22       And verify with "
C78D   F0 04      BEQ $C793      Identical?
C78F   C9 A0      CMP #$A0       NO—verify with 'shift space'
C791   D0 F0      BNE $C783      Identical?
C793²  A9 22      LDA #$22       YES—then replace with "
C795   9D B1 02   STA $02B1,X    (at end of data name)
C798   E8         INX            Set filename indicator to next
C799¹  E0 20      CPX #$20       byte and verify with end value
C79B   B0 0A      BCS $C7A7      End of file name reached?
C79D   A9 7F      LDA #$7F       NO—value f/bit 7(reverse)deleted
C79F   3D B1 02   AND $02B1,X    Get character of directory line
C7A2   9D B1 02   STA $02B1,X    And switch reverse off
C7A5   10 F1      BPL $C798      Always jump to $C798
C7A7¹  20 B5 C4   JSR $C4B5      Get next entry
C7AA   38         SEC            Flag for 'more entries'
C7AB   60         RTS            Back to calling routine
-----------------------------------------------------------------

[C73C/C7BD/C806]
Delete buffer for data name with empty character
C7AC   A0 1B      LDY #$1B       Length of directory line (27)
C7AE   A9 20      LDA #$20       Empty character as delete value
C7B0¹  99 B0 02   STA $02B0,Y    Delete buffer position
C7B3   88         DEY            Set buffer indicator to next byte
C7B4   D0 FA      BNE $C7B0      Buffer deleted?
C7B6   60         RTS            YES—return to calling routine
C7B7   20 19 F1   JSR $F119      Set pointer to BAM
C7BA   20 DF F0   JSR $F0DF      Read BAM from diskette
C7BD   20 AC C7   JSR $C7AC      Clear buffer for directory line
C7C0   A9 FF      LDA #$FF       Initialize
C7C2   85 6F      STA $6F        temporary storage
C7C4   A6 7F      LDX $7F        Write number of current drive
C7C6   8E 72 02   STX $0272      as two-byte value (as in block #)
C7C9   A9 00      LDA #$00       in directory buffer
C7CB   8D 73 02   STA $0273      Directory buffer
C7CE   A6 F9      LDX $F9        Get current buffer number
C7D0   BD E0 FE   LDA $FEE0,X    Get buffer address (high-byte)
C7D3   85 95      STA $95        and save it
C7D5   AD 88 FE   LDA $FE88      Take pos. of diskname as lo-byte
C7D8   85 94      STA $94        of buffer address
C7DA   A0 16      LDY #$16       Length of diskette name
C7DC   B1 94      LDA ($94),Y    Get character of name
C7DE   C9 A0      CMP #$A0       Compare with 'Shift Space'
C7E0   D0 0B      BNE $C7ED      Is diskette name at an end?
```

```
C7E2   A9 31      LDA #$31       YES—Dummy for test after ('1')
C7E4   2C         .byte $2C      Jump two bytes
C7E5¹  B1 94      LDA ($94),Y    Get char from directory entry
C7E7   C9 A0      CMP #$A0       Compare with 'Shift Space'
C7E9   D0 02      BNE $C7ED      Is entry to an end?
C7EB   A9 20      LDA #$20       YES—Transfer blank character into
C7ED²  99 B3 02   STA $02B3,Y    buffer and set buffer pointer
C7F0   88         DEY            to next byte
C7F1   10 F2      BPL $C7E5      End of buffer reached?
C7F3   A9 12      LDA #$12       YES—Code for 'Reverse On' on
C7F5   8D B1 02   STA $02B1      Set line beginning in buffer
C7F8   A9 22      LDA #$22       Put quotation marks before and x
C7FA   8D B2 02   STA $02B2      after the diskette
C7FD   8D C3 02   STA $02C3      name
C800   A9 20      LDA #$20       Write space in
C802   8D C4 02   STA $02C4      buffer
C805   60         RTS            return from this subroutine
-------------------------------------------------------------------
[C6F2/C707]
Set up closing line with 'Blocks free.'
C806   20 AC C7   JSR $C7AC      Clear buffer for directory line
C809   A0 0B      LDY #$0B       Set linelength
C80B¹  B9 17 C8   LDA $C817,Y    Get char from 'Blocks Free'string
C80E   99 B1 02   STA $02B1,Y    and write into buffer
C811   88         DEY            Set buffer pointer to next byte
C812   10 F7      BPL $C80B      Line ready?
C814   4C 4D EF   JMP $EF4D      YES—Get number of free blocks
-------------------------------------------------------------------
C817   42 4C 4F 43 4B 53 20      'BLOCKS '
C81E   46 52 45 45 2E            'FREE.'
-------------------------------------------------------------------
[Origin through routine C146]
Routine for Scratch command
C823   20 98 C3   JSR $C398      Chk if cmd is limited/filetype
C826   20 20 C3   JSR $C320      Get drive # from command string
C829   20 CA C3   JSR $C3CA      Initialize drive
C82C   A9 00      LDA #$00       Set back counter for number of
C82E   85 86      STA $86        deleted files
C830   20 9D C4   JSR $C49D      Get first file entry
C833   30 3D      BMI $C872      Entry found?
C835¹  20 B7 DD   JSR $DDB7      YES—Test file for validity
C838   90 33      BCC $C86D      Has file been closed properly?
C83A   A0 00      LDY #$00       YES—Pointer to filetype position
C83C   B1 94      LDA ($94),Y    Get filetype from directory
C83E   29 40      AND #$40       Test bit6 as flag f/scratch prot.
C840   D0 2B      BNE $C86D      Is the file ready for scratching?
```

```
C842   20 B6 C8   JSR $C8B6      NO-Delete entry
C845   A0 13      LDY #$13       Set pointer to side-sector entry
C847   B1 94      LDA ($94),Y    Get track # of first side-sector
C849   F0 0A      BEQ $C855      Side-sector onhand?
C84B   85 80      STA $80        YES-Save track number and
C84D   C8         INY            get corresponding
C84E   B1 94      LDA ($94),Y    sector number as entry
C850   85 81      STA $81        and Save it
C852   20 7D C8   JSR $C87D      Pursue and free up blocks
C855   AD 78 02   LDA $0278      Entry number
C858   A9 20      LDA #$20       Check Flag for 'File not closed'
C85A   35 E7      AND $E7,X      in filetype identifier
C85C   D0 0D      BNE $C86B      Is entry a '*' file?
C85E   BD 80 02   LDA $0280,X    NO-Set track of first sector
C861   85 80      STA $80        as current track number
C863   BD 85 02   LDA $0285,X    Take up sector number of
C866   85 81      STA $81        file data
C868   20 7D C8   JSR $C87D      Follow and free up file blocks
C86B1  E6 86      INC $86        Increment # of scatched files
C86D2  20 8B C4   JSR $C48B      Get next file entry
C870   10 C3      BPL $C835      Found it?
C8721  A5 86      LDA $86        NO-Give # of deleted files to
C874   85 80      STA $80        return message
C876   A9 01      LDA #$01       Number of return message
C878   A0 00      LDY #$00       Value for sector number
C87A   4C A3 C1   JMP $C1A3      Display '01 Files Scratched'
----------------------------------------------------------------
[C852/C868/DC1B]
Pursue sectors onhand and free up in BAM
C87D   20 5F EF   JSR $EF5F      Free up first and current blocks
C880   20 75 D4   JSR $D475      Read next sector
C883   20 19 F1   JSR $F119      Get number of BAM channel
C886   B5 A7      LDA $A7,X      Get number of 2nd buffer
C888   C9 FF      CMP #$FF       Compare with 'Buffer free'
C88A   F0 08      BEQ $C894      Is buffer allocated?
C88C   AD F9 02   LDA $02F9      NO-Set flag in pointer for
C88F   09 40      ORA #$40       'BAM illegal for writing
C891   8D F9 02   STA $02F9      to diskette'
C8942  A9 00      LDA #$00       Set buffer pointer
C896   20 C8 D4   JSR $D4C8      to beginning of sector
C899   20 56 D1   JSR $D156      Get byte from sector and save
C89C   85 80      STA $80        track of next sector
C89E   20 56 D1   JSR $D156      Get byte from sector and set
C8A1   85 81      STA $81        number of next sector
C8A3   A5 80      LDA $80        Get track number of next sector
C8A5   D0 06      BNE $C8AD      Is the current sector the last?
```

```
C8A7  20 F4 EE   JSR $EEF4       YES—Write BAM to diskette again
C8AA  4C 27 D2   JMP $D227       Re-close channel and end
C8AD[1] 20 5F EF JSR $EF5F       Free up sector in BAM
C8B0  20 4D D4   JSR $D44D       Read next sector and
C8B3  4C 94 C8   JMP $C894       continue
-----------------------------------------------------------------------
[C842/D8D3/EDDF]
File entry in filetype of directory marked as scratched
C8B6  A0 00      LDY #$00        Set buffer pointer to filetype
C8B8  98         TYA             Take value for 'DEL' filetype
C8B9  91 94      STA ($94),Y     in entry and adjust
C8BB  20 5E DE   JSR $DE5E       directory
C8BE  4C 99 D5   JMP $D599       Wait until writing is done
-----------------------------------------------------------------------
[C909/Origin through routine C146]
Backup command routine (not possible with single drive)
C8C1  A9 31      LDA #$31        Display
C8C3  4C C8 C1   JMP $C1C8       '31 Syntax Error' message & rturn
-----------------------------------------------------------------------
[A780]
Routine for 1541 New command (format diskette)
C8C6  A9 4C      LDA #$4C        JMP-pointer for format routine
C8C8  8D 00 06   STA $0600       in buffer address $0600-$0602
C8CB  A9 C7      LDA #$C7        set for disk controller,
C8CD  8D 01 06   STA $0601       ($FAC7), re-calling its own sub-
C8D0  A9 FA      LDA #$FA        program for every
C8D2  8D 02 06   STA $0602       new track
C8D5  A9 03      LDA #$03        Number of buffer used
C8D7  20 D3 D6   JSR $D6D3       Track / sector number to jobloop
C8DA  A5 7F      LDA $7F         Get current drive number
C8DC  09 E0      ORA #$E0        Tie in jobcode for buffer program
C8DE  85 03      STA $03         (jump to pointer)
C8E0[1] A5 03    LDA $03         and get return message
C8E2  30 FC      BMI $C8E0       Wait until diskette is formatted
C8E4  C9 02      CMP #$02        Compare return message with 'OK'
C8E6  90 07      BCC $C8EF       Format ended error-free?
C8E8  A9 03      LDA #$03        NO—Error number for 'File'
C8EA  A2 00      LDX #$00        Go to buffer 0 and
C8EC  4C 0A E6   JMP $E60A       display message
C8EF[1] 60       RTS             Return from this subroutine
-----------------------------------------------------------------------
```

```
[Origin through C146]
COPY command routine (file copier)
C8F0  A9 E0     LDA #$E0       Set up all buffers
C8F2  8D 4F 02  STA $024F      in bit library
C8F5  20 D1 F0  JSR $F0D1      Set track / sector number for BAM
C8F8  20 19 F1  JSR $F119      Determine buffer number of BAM
C8FB  A9 FF     LDA #$FF       BAM buffer marked with
C8FD  95 A7     STA $A7,X      'free' identifier
C8FF  A9 0F     LDA #$0F       Free up all channels for cor-
C901  8D 56 02  STA $0256      responding bit library
C904  20 E5 C1  JSR $C1E5      Look for ':' in command string
C907  D0 03     BNE $C90C      Found it?
C909  4C C1 C8  JMP $C8C1      NO—error messge:'31 Syntax Error'
C90C1 20 F8 C1  JSR $C1F8      Work with input string
C90F  20 20 C3  JSR $C320      Get / set drive number
C912  AD 8B 02  LDA $028B      Get command syntax flag and get
C915  29 55     AND #$55       flags for filenames
C917  D0 0F     BNE $C928      Are several filenames onhand?
C919  AE 7A 02  LDX $027A      YES—Pos. of command target name
C91C  BD 00 02  LDA $0200,X    Get character from filename
C91F  C9 2A     CMP #$2A       Check for '*' wildcard
C921  D0 05     BNE $C928      Wildcard onhand?
C9231 A9 30     LDA #$30       YES—Display
C925  4C C8 C1  JMP $C1C8      '30 Syntax Error' message
C9282 AD 8B 02  LDA $028B      Get command syntax flag
C92B  29 D9     AND #$D9       Test flag for wildcard
C92D  D0 F4     BNE $C923      Are wildcards onhand?
C92F  4C 52 C9  JMP $C952      NO—Copy file
-------------------------------------------------------------------
[Routine not used in DOS]
Initialize Backup- command pointer (Command not onhand)
C932  A9 00     LDA #$00       Clear pointer:
C934  8D 58 02  STA $0258      Length of a record
C937  8D 8C 02  STA $028C      Number of disk accesses
C93A  8D 80 02  STA $0280      Track number of target file
C93D  8D 81 02  STA $0281      Track number of source file
C940  A5 E3     LDA $E3        Value for standard drive
C942  29 01     AND #$01       Limit declaration to bit 0 and
C944  85 7F     STA $7F        pointer for current drive
C946  09 01     ORA #$01       Set back number of current
C948  8D 91 02  STA $0291      directory sector
C94B  AD 7B 02  LDA $027B      Copy position of 2nd parameter in
C94E  8D 7A 02  STA $027A      first place
C951  60        RTS            Return from this subroutine
-------------------------------------------------------------------
```

```
[C92F] Copy file(s)
C952  20 4F C4  JSR $C44F     Look for file entry in directory
C955  AD 78 02  LDA $0278     Number of source files named
C958  C9 03     CMP #$03      Individual files
C95A  90 45     BCC $C9A1     Less than 3 files named?
C95C  A5 E2     LDA $E2       YES—Compare drive # of targetfile
C95E  C5 E3     CMP $E3       with sourcefile drive
C960  D0 3F     BNE $C9A1     Copy only one diskette?
C962  A5 DD     LDA $DD       YES—Compare # of target files in
C964  C5 DE     CMP $DE       directory with source files
C966  D0 39     BNE $C9A1     Identical?
C968  A5 D8     LDA $D8       YES—Test # of appropriate dir
C96A  C5 D9     CMP $D9       sectors with those on sourcefile
C96C  D0 33     BNE $C9A1     Should entry be overwritten?
C96E  20 CC CA  JSR $CACC     YES—Look for file entry in dir.
C971  A9 01     LDA #$01      Set pointer to first
C973  8D 79 02  STA $0279     filename
C976  20 FA C9  JSR $C9FA     Open file for reading and
C979  20 25 D1  JSR $D125     get filetype
C97C  F0 04     BEQ $C982     Is file entry a relative file?
C97E  C9 02     CMP #$02      NO—Check for 'PRG' identifier
C980  D0 05     BNE $C987     Identical?
C982¹ A9 64     LDA #$64      Display '64 File Type Mismatch'
C984  20 C8 C1  JSR $C1C8     message
C987¹ A9 12     LDA #$12      Set
C989  85 83     STA $83       internal write channel (18)
C98B  AD 3C 02  LDA $023C     Transfer # of allocated internal
C98E  8D 3D 02  STA $023D     channel in read channel
C991  A9 FF     LDA #$FF      Set 'Channel free' flag
C993  8D 3C 02  STA $023C     in table
C996  20 2A DA  JSR $DA2A     Copy 1st sourcefile to targetfile
C999  A2 02     LDX #$02      Pointer of second filename to
C99B  20 B9 C9  JSR $C9B9     next file
C99E  4C 94 C1  JMP $C194     End command; display 'OK'
C9A1⁴ 20 A7 C9  JSR $C9A7     Copy file
C9A4  4C 94 C1  JMP $C194     End command; display 'OK'
---------------------------------------------------------------
[C9A1] Copy individual files
C9A7  20 E7 CA  JSR $CAE7     See if entry already exists
C9AA  A5 E2     LDA $E2       Get drive indicator of targetfile
C9AC  29 01     AND #$01      and take on as number of
C9AE  85 7F     STA $7F       current drive
C9B0  20 86 D4  JSR $D486     Open internal channel for writing
C9B3  20 E4 D6  JSR $D6E4     Enter target file in directory
C9B6  AE 77 02  LDX $0277     Take number of target names as
---------------------------------------------------------------
```

```
[C99B/C9F1]
Copy multiple files
C9B9  8E 79 02   STX $0279     number of source names
C9BC  20 FA C9   JSR $C9FA     Read directory
C9BF  A9 11      LDA #$11      16 (# of internal read channel)
C9C1  85 83      STA $83       set as current secondary address
C9C3  20 EB D0   JSR $D0EB     Open channel
C9C6  20 25 D1   JSR $D125     Get filetype of entry
C9C9  D0 03      BNE $C9CE     Is file a REL file?
C9CB  20 53 CA   JSR $CA53     YES—Copy relative file
C9CE1 A9 08      LDA #$08      Set flag for last character
C9D0  85 F8      STA $F8       (EOI) and conclude
C9D2  4C D8 C9   JMP $C9D8     copy procedure
C9D51 20 9B CF   JSR $CF9B     Write byte in target file
C9D81 20 35 CA   JSR $CA35     Get byte from source file
C9DB  A9 80      LDA #$80      Test EOI (last character)
C9DD  20 A6 DD   JSR $DDA6     flag
C9E0  F0 F3      BEQ $C9D5     Is flag set?
C9E2  20 25 D1   JSR $D125     YES—Get filetype
C9E5  F0 03      BEQ $C9EA     Is file entry a relative file?
C9E7  20 9B CF   JSR $CF9B     NO—Write byte in file
C9EA1 AE 79 02   LDX $0279     Compare number of target files
C9ED  E8         INX           with number of
C9EE  EC 78 02   CPX $0278     source files
C9F1  90 C6      BCC $C9B9     Any more files given?
C9F3  A9 12      LDA #$12      Set write channel number (18)
C9F5  85 83      STA $83       as current secodnary address
C9F7  4C 02 DB   JMP $DB02     Close file and channel
-----------------------------------------------------------------
[C976/C9BC]
Open channel for file reading
C9FA  AE 79 02   LDX $0279     Get number of filename
C9FD  B5 E2      LDA $E2,X     Establish corresponding drive #
C9FF  29 01      AND #$01      and save as
CA01  85 7F      STA $7F       current drive
CA03  AD 85 FE   LDA $FE85     Set # of directory track (18)
CA06  85 80      STA $80       as current track
C95C  A5 E2      LDA $E2       YES—Compare drive # of targetfile
C95E  C5 E3      CMP $E3       with sourcefile drive
C960  D0 3F      BNE $C9A1     Copy only one diskette?
C962  A5 DD      LDA $DD       YES—Compare # of targetfiles in
C964  C5 DE      CMP $DE       directory with source files
C966  D0 39      BNE $C9A1     Identical?
C968  A5 D8      LDA $D8       YES—Test # of matching directory
C96A  C5 D9      CMP $D9       sector against sourcefile
C96C  D0 33      BNE $C9A1     Should entry be overwritten?
```

```
C96E   20 CC CA   JSR $CACC        YES—Look for file entry in dir.
C971   A9 01      LDA #$01         Set pointer to
C973   8D 79 02   STA $0279        first filename
C976   20 FA C9   JSR $C9FA        Open file for reading
C979   20 25 D1   JSR $D125        and get filetype
C97C   F0 04      BEQ $C982        Is file entry a relative file?
C97E   C9 02      CMP #$02         NO—Test for 'PRG' identifier
C980   D0 05      BNE $C987        Identical?
C982¹  A9 64      LDA #$64         YES—Display
C984   20 C8 C1   JSR $C1C8        '64 File Type Mismatch' message
C987¹  A9 12      LDA #$12         Set internal read channel
C989   85 83      STA $83          (18)
C98B   AD 3C 02   LDA $023C        Transfer # assigned to internal
C98E   8D 3D 02   STA $023D        channel in read channel
C991   A9 FF      LDA #$FF         Set 'Channel free' value
C993   8D 3C 02   STA $023C        in table
C996   20 2A DA   JSR $DA2A        Copy 1st sourcefile to targetfile
C999   A2 02      LDX #$02         Pointer to second filename
C99B   20 B9 C9   JSR $C9B9        Attach next file
C99E   4C 94 C1   JMP $C194        End command and display 'OK'
C9A1⁴  20 A7 C9   JSR $C9A7        Copy file
C9A4   4C 94 C1   JMP $C194        End command and display 'OK'
------------------------------------------------------------------------
[C9A1]
Copy single files
C9A7   20 E7 CA   JSR $CAE7        Test whether entry already exists
C9AA   A5 E2      LDA $E2          Get drive indicator of targetfile
C9AC   29 01      AND #$01         and take on as number of
C9AE   85 7F      STA $7F          current drive
C9B0   20 86 D4   JSR $D486        Open internal channel for writing
C9B3   20 E4 D6   JSR $D6E4        Enter target file in directory
C9B6   AE 77 02   LDX $0277        Number of target names (1)
------------------------------------------------------------------------
[C99B/C9F1] Copy several files
C9B9   8E 79 02   STX $0279        Number of sourcefiles
C9BC   20 FA C9   JSR $C9FA        Read directory
C9BF   A9 11      LDA #$11         Set 16 (number of internal read
C9C1   85 83      STA $83          channels) as current 2ndary addr
C9C3   20 EB D0   JSR $D0EB        Open channel
C9C6   20 25 D1   JSR $D125        Get filetype from entry
C9C9   D0 03      BNE $C9CE        Is it a relative file?
C9CB   20 53 CA   JSR $CA53        YES—Copy relative file
C9CE¹  A9 08      LDA #$08         Set 'last character' (EOI)
C9D0   85 F8      STA $F8          flag
C9D2   4C D8 C9   JMP $C9D8        and conclude copy process
C9D5¹  20 9B CF   JSR $CF9B        Write byte in target file
```

```
C9D8¹ 20 35 CA   JSR $CA35     Get byte from sourcefile
C9DB  A9 80      LDA #$80      Test EOI (last character)
C9DD  20 A6 DD   JSR $DDA6     flag
C9E0  F0 F3      BEQ $C9D5     Is flag set?
C9E2  20 25 D1   JSR $D125     YES—Get filetype
C9E5  F0 03      BEQ $C9EA     Is file entry a REL file?
C9E7  20 9B CF   JSR $CF9B     NO—write byte in file
C9EA¹ AE 79 02   LDX $0279     Compare sourcefile number
C9ED  E8         INX           with number
C9EE  EC 78 02   CPX $0278     of sourcefiles
C9F1  90 C6      BCC $C9B9     Any more files left?
C9F3  A9 12      LDA #$12      Set write channel number (8)
C9F5  85 83      STA $83       as current secondary address
C9F7  4C 02 DB   JMP $DB02     Close file and channel
------------------------------------------------------------
[C976/C9BC]
Open channel to read file
C9FA  AE 79 02   LDX $0279     Get filename number
C9FD  B5 E2      LDA $E2,X     Determine corresponding
C9FF  29 01      AND #$01      drive number and save
CA01  85 7F      STA $7F       as current drive number
CA03  AD 85 FE   LDA $FE85     Set up # of directory track (18)
CA06  85 80      STA $80       as current track
CA08  B5 D8      LDA $D8,X     Determine sector of entry; set as
CA0A  85 81      STA $81       current sector
CA0C  20 75 D4   JSR $D475     Read sector in buffer
CA0F  AE 79 02   LDX $0279     # of file identifier in command
CA12  B5 DD      LDA $DD,X     Get correct pointer frm directory
CA14  20 C8 D4   JSR $D4C8     position and set buffer pointer
CA17  AE 79 02   LDX $0279     File indication number of command
CA1A  B5 E7      LDA $E7,X     Get corrspondng filetype identifr
CA1C  29 07      AND #$07      and get filetype from that;
CA1E  8D 4A 02   STA $024A     save it
CA21  A9 00      LDA #$00      Clear file record length
CA23  8D 58 02   STA $0258     pointer
CA26  20 A0 D9   JSR $D9A0     Open file for reading
CA29  A0 01      LDY #$01      Set puffer pointer
CA2B  20 25 D1   JSR $D125     Get filetype
CA2E  F0 01      BEQ $CA31     Is file a relative file?
CA30  C8         INY           NO—Buffer pointer to next byte
CA31¹ 98         TYA           (track number)
CA32  4C C8 D4   JMP $D4C8     Initialize buffer pointer
------------------------------------------------------------
```

```
[C9D8/E81B/E839]
Read a byte from file
CA35  A9 11      LDA #$11      Set internal channel number
CA37  85 83      STA $83       for reading
CA39  20 9B D3   JSR $D39B     Read a byte
CA3C  85 85      STA $85       and save it
CA3E  A6 82      LDX $82       Get channel number and determine
CA40  B5 F2      LDA $F2,X     channel status
CA42  29 08      AND #$08      Detrmin bitflg f/'last byte'(EOI)
CA44  85 F8      STA $F8       and save it
CA46  D0 0A      BNE $CA52     End of file?
CA48  20 25 D1   JSR $D125     NO—Get filetype
CA4B  F0 05      BEQ $CA52     Is it a relative file?
CA4D  A9 80      LDA #$80      NO—
CA4F  20 97 DD   JSR $DD97     Set all corresponding flags
CA52² 60         RTS           Return from this subroutine
------------------------------------------------------------------
[C9CB]
Copy relative file
CA53  20 D3 D1   JSR $D1D3     Set current drive number
CA56  20 CB E1   JSR $E1CB     Get position of last record
CA59  A5 D6      LDA $D6       Save position in
CA5B  48         PHA           side-sector;
CA5C  A5 D5      LDA $D5       hold number of corresponding
CA5E  48         PHA           side-sector
CA5F  A9 12      LDA #$12      Set internal channel for
CA61  85 83      STA $83       writing
CA63  20 07 D1   JSR $D107     Open channel
CA66  20 D3 D1   JSR $D1D3     Set current drive number
CA69  20 CB E1   JSR $E1CB     Get position of last side-sector
CA6C  20 9C E2   JSR $E29C     and read sector in buffer
CA6F  A5 D6      LDA $D6       Save current pointer at position
CA71  85 87      STA $87       in side-sector
CA73  A5 D5      LDA $D5       Save number of
CA75  85 86      STA $86       side-sector
CA77  A9 00      LDA #$00      Clear pointer:
CA79  85 88      STA $88       temporary memory
CA7B  85 D4      STA $D4       Pointer to beginning of record
CA7D  85 D7      STA $D7       Pointer to position in record
CA7F  68         PLA           Get number of last side-sector
CA80  85 D5      STA $D5       and set it
CA82  68         PLA           Get # of last record entry in
CA83  85 D6      STA $D6       side-sector; save it
CA85  4C 3B E3   JMP $E33B     Actualize side-sectors
------------------------------------------------------------------
```

```
[Origin at routine C146]
Routine for Rename command
CA88   20 20 C3   JSR $C320      Get drive number
CA8B   A5 E3      LDA $E3        Establish # of standard drive
CA8D   29 01      AND #$01       and
CA8F   85 E3      STA $E3        reset
CA91   C5 E2      CMP $E2        Compare with last drive number
CA93   F0 02      BEQ $CA97      Must drive be changed?
CA95   09 80      ORA #$80       YES—Set bitflag for search of
CA97¹  85 E2      STA $E2        both drives
CA99   20 4F C4   JSR $C44F      Search for new name in directory
CA9C   20 E7 CA   JSR $CAE7      Name already there?
CA9F   A5 E3      LDA $E3        Establish # of standard drive and
CAA1   29 01      AND #$01       take on as number of current
CAA3   85 7F      STA $7F        drive
CAA5   A5 D9      LDA $D9        Set number of directory
CAA7   85 81      STA $81        sector
CAA9   20 57 DE   JSR $DE57      and read sector into buffer;
CAAC   20 99 D5   JSR $D599      Wait until command is executed
CAAF   A5 DE      LDA $DE        Set directory entry pointer to
CAB1   18         CLC            starting position
CAB2   69 03      ADC #$03       of filenames in directory
CAB4   20 C8 D4   JSR $D4C8      Establish buffer pointer
CAB7   20 93 DF   JSR $DF93      Get and save number of
CABA   A8         TAY            current buffer
CABB   AE 7A 02   LDX $027A      Position of new name in command
CABE   A9 10      LDA #$10       Max. length of filename
CAC0   20 6E C6   JSR $C66E      Names in buffer frm commnd string
CAC3   20 5E DE   JSR $DE5E      Rewrite directory sector
CAC6   20 99 D5   JSR $D599      and wait until executed
CAC9   4C 94 C1   JMP $C194      Prepare return message and end
-------------------------------------------------------------
[C96E/CAE7] See if file entry is onhand
CACC   A5 E8      LDA $E8        Get filetype of 2nd name &
CACE   29 07      AND #$07       isolate type identifier
CAD0   8D 4A 02   STA $024A      Save as current filetype
CAD3   AE 78 02   LDX $0278      Get starting position of filename
CAD6¹  CA         DEX            in command string
CAD7   EC 77 02   CPX $0277      Compare w/start of command string
CADA   90 0A      BCC $CAE6      More characters in filenames?
CADC   BD 80 02   LDA $0280,X    YES—Get sector number of file
CADF   D0 F5      BNE $CAD6      Was that the last sector?
CAE1   A9 62      LDA #$62       YES—Display
CAE3   4C C8 C1   JMP $C1C8      '62 File Not Found'
CAE6¹  60         RTS            Return from this subroutine
-------------------------------------------------------------
```

```
[C9A7/CA9C]
Compare with two filenames
CAE7   20 CC CA   JSR $CACC       File in directory onhand?
CAEA1  BD 80 02   LDA $0280,X     Get number of first file sector
CAED   F0 05      BEQ $CAF4       Is sector onhand?
CAEF   A9 63      LDA #$63        YES—Display
CAF1   4C C8 C1   JMP $C1C8       '63 File exist'
CAF41  CA         DEX             Go to next name
CAF5   10 F3      BPL $CAEA       Was that the last filename?
CAF7   60         RTS             YES—Return from this subroutine
------------------------------------------------------------------
[Origin at routine C146]
Memory-command routine
CAF8   AD 01 02   LDA $0201       Get second character of command
CAFB   C9 2D      CMP #$2D        Compare with '-'
CAFD   D0 4C      BNE $CB4B       Identical?
CAFF   AD 03 02   LDA $0203       YES—Then get fourth character and
CB02   85 6F      STA $6F         set as memory address (low-byte)
CB04   AD 04 02   LDA $0204       Get fifth character and save as
CB07   85 70      STA $70         memory address (high-byte)
CB09   A0 00      LDY #$00        Clear buffer pointer
CB0B   AD 02 02   LDA $0202       Get third character of command
CB0E   C9 52      CMP #$52        and compare with 'R'
CB10   F0 0E      BEQ $CB20       Should Read command be performed?
CB12   20 58 F2   JSR $F258       NO—Call has no function (RTS)
CB15   C9 57      CMP #$57        Compare with 'W'
CB17   F0 37      BEQ $CB50       Should Write commnd be performed?
CB19   C9 45      CMP #$45        NO—Compare with 'E'
CB1B   D0 2E      BNE $CB4B       Should program be performed?
CB1D   6C 6F 00   JMP ($006F)     YES—Start program
------------------------------------------------------------------
[CB10]
Memory-Read command ('M-R'); Read byte from memory
CB20   B1 6F      LDA ($6F),Y     Get byte from given address
CB22   85 85      STA $85         and save it
CB24   AD 74 02   LDA $0274       Get length of command string and
CB27   C9 06      CMP #$06        compare with maximum length
CB29   90 1A      BCC $CB45       Is the string smaller?
CB2B   AE 05 02   LDX $0205       NO—Get # of bytes to be read and
CB2E   CA         DEX             adjust (one already read)
CB2F   F0 14      BEQ $CB45       Read any more bytes from memory?
CB31   8A         TXA             YES—Balance pointer
CB32   18         CLC             with starting address and then
CB33   65 6F      ADC $6F         compute end address of this range
CB35   E6 6F      INC $6F         Increment pointer to current byte
CB37   8D 49 02   STA $0249       Save ending address (low-byte)
```

```
CB3A   A5 6F       LDA $6F          Take pointer to current memory;
CB3C   85 A5       STA $A5          use as pointer to error message
CB3E   A5 70       LDA $70          buffer for routine that is
CB40   85 A6       STA $A6          to follow ($D43A)
CB42   4C 43 D4    JMP $D443        Set first byte and output flag
CB45²  20 EB D0    JSR $D0EB        Seek out and open channel
CB48   4C 3A D4    JMP $D43A        Output more bytes
CB4B²  A9 31       LDA #$31         Display
CB4D   4C C8 C1    JMP $C1C8        '31 Syntax Error'
-----------------------------------------------------------------
[CB17/CB59] Memory-Write command ('M-W'); Write in memory
CB50   B9 06 02    LDA $0206,Y      Get byte val from command string
CB53   91 6F       STA ($6F),Y      and write into memory
CB55   C8          INY              Turn buffer pointer to next byte
CB56   CC 05 02    CPY $0205        Compare with value for 'End'
CB59   90 F5       BCC $CB50        Take any more bytes?
CB5B   60          RTS              NO—Return from this subroutine
-----------------------------------------------------------------
[Origin at routine C146]
User-command ('UX'); Start program in DOS buffer
CB5C   AC 01 02    LDY $0201        Get second char of command and
CB5F   C0 30       CPY #$30         compare with '0'
CB61   D0 09       BNE $CB6C        Identical?
-----------------------------------------------------------------
[EBBC] Execute User-command
CB63   4C 26 80    JMP $8030        YES—Read User-0 command
CB66   EA          NOP              Unused space
CB67   EA          NOP              left by modifying
CB68   EA          NOP              ROM User-routine
CB69   EA          NOP              in 1541 drive
CB6A   EA          NOP              to
CB6B   EA          NOP              1571 User-routine
CB6C¹  20 72 CB    JSR $CB72        Set address and execute program
CB6F   4C 94 C1    JMP $C194        End program by 'RTS'
CB72¹  88          DEY              Convert ASCII number of command
CB73   98          TYA              into binary
CB74   29 0F       AND #$0F         number; double it
CB76   0A          ASL A            (address is 2-byte pointer)
CB77   A8          TAY              and save it
CB78   B1 6B       LDA ($6B),Y      Get address belonging to command
CB7A   85 75       STA $75          (low-byte) and save it
CB7C   C8          INY              Pointer to next byte of address
CB7D   B1 6B       LDA ($6B),Y      Get high-byte of starting address
CB7F   85 76       STA $76          and save it
CB81   4C 2D AA    JMP $AA2D        Start program
-----------------------------------------------------------------
```

```
[D819]
'#'-command; Open direct access channel
CB84   AD 8E 02   LDA $028E       Set drive number of last job
CB87   85 7F      STA $7F         as current drive
CB89   A5 83      LDA $83         Get channel number and
CB8B   48         PHA             save it
CB8C   20 3D C6   JSR $C63D       Initialize drive
CB8F   68         PLA             Re-set channel
CB90   85 83      STA $83         number
CB92   AE 74 02   LDX $0274       Compare length of command
CB95   CA         DEX             string with 1
CB96   D0 0D      BNE $CBA5       Is a desired buffer given?
CB98   A9 01      LDA #$01        NO—Number of buffers needed
CB9A   20 E2 D1   JSR $D1E2       Set up buffer and channel
CB9D   4C F1 CB   JMP $CBF1       Pointer and table initialization
CBA0³  A9 70      LDA #$70        Display error message --
CBA2   4C C8 C1   JMP $C1C8       '70 No Channel'
CBA5¹  A0 01      LDY #$01        Pointer to position in buffer
CBA7   20 7C CC   JSR $CC7C       Get byte from command string
CBAA   AE 85 02   LDX $0285       Get buffer number and compare
CBAD   E0 05      CPX #$05        with maximum buffer
CBAF   B0 EF      BCS $CBA0       Is the given number allowed (<5)?
CBB1   A9 00      LDA #$00        YES—Clear temporary
CBB3   85 6F      STA $6F         storage in
CBB5   85 70      STA $70         zeropage
CBB7   38         SEC             Shift 'Buffer occupied'
CBB8¹  26 6F      ROL $6F         in temporary
CBBA   26 70      ROL $70         memory
CBBC   CA         DEX             Buffer number
CBBD   10 F9      BPL $CBB8       Is flag in the correct position?
CBBF   A5 6F      LDA $6F         Compare computed buffer set-up
CBC1   2D 4F 02   AND $024F       with bit table
CBC4   D0 DA      BNE $CBA0       Is buffer already occupied?
CBC6   A5 70      LDA $70         NO—Test buffer numbers 8-15
CBC8   2D 50 02   AND $0250       (only on CBM3030-CBM8250)
CBCB   D0 D3      BNE $CBA0       Is buffer free?
CBCD   A5 6F      LDA $6F         YES—Take buffer bit
CBCF   0D 4F 02   ORA $024F       in bit table and
CBD2   8D 4F 02   STA $024F       set up buffer
CBD5   A5 70      LDA $70         The same goes for buffers 8-15
CBD7   0D 50 02   ORA $0250       (there can only be 5 buffers at
CBDA   8D 50 02   STA $0250       a time)
CBDD   A9 00      LDA #$00        Set number of buffers to 1
CBDF   20 E2 D1   JSR $D1E2       and set up buffer and channel
CBE2   A6 82      LDX $82         Current channel number
CBE4   AD 85 02   LDA $0285       Current sector number
```

```
CBE7   95 A7      STA $A7,X       Arrange in channel sector table
CBE9   AA         TAX             Adjust pointer
CBEA   A5 7F      LDA $7F         Give current drive number
CBEC   95 00      STA $00,X       as
CBEE   9D 5B 02   STA $025B,X     jobcode
CBF1¹  A6 83      LDX $83         Determine secondary address and
CBF3   BD 2B 02   LDA $022B,X     get pre-arranged internal channel
CBF6   09 40      ORA #$40        Identify channel
CBF8   9D 2B 02   STA $022B,X     as in an 'active' state
CBFB   A4 82      LDY $82         Current channel number
CBFD   A9 FF      LDA #$FF        Arrange number of data to be sent
CBFF   99 44 02   STA $0244,Y     over channel
CC02   A9 89      LDA #$89        Free up channel for
CC04   99 F2 00   STA $00F2,Y     reading/writing
CC07   B9 A7 00   LDA $00A7,Y     Get buffer number
CC0A   99 3E 02   STA $023E,Y     Set as characters to be given
CC0D   0A         ASL A           Double number
CC0E   AA         TAX             (table has 2-byte values)
CC0F   A9 01      LDA #$01        Set buffer pointer to beginning
CC11   95 99      STA $99,X       of buffer
CC13   A9 0E      LDA #$0E        note directory access identifier
CC15   99 EC 00   STA $00EC,Y     in filetype table
CC18   4C 94 C1   JMP $C194       Send acknowledgement and end it
------------------------------------------------------------------
[Origin at Routine C146]
Routine for Block command
CC1B   A0 00      LDY #$00        Set start position in input buffr
CC1D   A2 00      LDX #$00        Clear pointer to # of parameters
CC1F   A9 2D      LDA #$2D        Set '-' as character to be sought
CC21   20 68 C2   JSR $C268       Process input string
CC24   D0 0A      BNE $CC30       Character found?
CC26¹  A9 31      LDA #$31        NO—Display
CC28   4C C8 C1   JMP $C1C8       '31 Syntax Error'
CC2B²  A9 30      LDA #$30        Display
CC2D   4C C8 C1   JMP $C1C8       '30 Syntax Error'
CC30¹  8A         TXA             Number of parameters found
CC31   D0 F8      BNE $CC2B       Any other givens found?
CC33   A2 05      LDX #$05        YES—Set pointer in input buffer
CC35   B9 00 02   LDA $0200,Y     Get third character from buffer
CC38¹  DD 5D CC   CMP $CC5D,X     and compare with Block command
CC3B   F0 05      BEQ $CC42       Is there a Block command?
CC3D   CA         DEX             NO—Set pointer to next command
CC3E   10 F8      BPL $CC38       Already compared w/other cmds?
CC40   30 E4      BMI $CC26       YES—Jump to $CC26
CC42¹  8A         TXA             Block command number
```

```
CC43   09 80      ORA #$80         Save 'Extended command'
CC45   8D 2A 02   STA $022A        flag
CC48   20 6F CC   JSR $CC6F        Get command parameters & test
CC4B   AD 2A 02   LDA $022A        Repeat command number and
CC4E   0A         ASL A            double it
CC4F   AA         TAX              (2-byte pointers in addr. table)
CC50   BD 64 CC   LDA $CC64,X      Get / save starting address of
CC53   85 70      STA $70          command (low-byte)
CC55   BD 63 CC   LDA $CC63,X      Get high-byte and take
CC58   85 6F      STA $6F          up in pointer
CC5A   6C 6F 00   JMP ($006F)      Start Block command
-------------------------------------------------------------
[CC38]  Command codes of Block command
CC5D   41 46 52 57 45 50           'A' , 'F' , 'R' , 'W' , 'E' , 'P'
-------------------------------------------------------------
[CC50/CC55]  Starting addresses of Block command routines
CC63   03 CD                       $CD03  B-A command
CC65   F5 CC                       $CCF5  B-F command
CC67   56 CD                       $CD56  B-R command
CC69   73 CD                       $CD73  B-W command
CC6B   A3 CD                       $CDA3  B-E command
CC6D   BD CD                       $CDBD  B-P command
-------------------------------------------------------------
[CC48/CD5F/CD97]
Get/set Block command parameters
CC6F   A0 00      LDY #$00         Start. pos.:commandstring search
CC71   A2 00      LDX #$00         Clear number of found parameters
CC73   A9 3A      LDA #$3A         Set ':' as character for search
CC75   20 68 C2   JSR $C268        and search in input buffer
CC78   D0 02      BNE $CC7C        Character found?
CC7A   A0 03      LDY #$03         NO—Buffer pointer to 4th char
-------------------------------------------------------------
[CBA7/CC78/CC8F]
Test Block command parameters
CC7C   B9 00 02   LDA $0200,Y      and get character
CC7F   C9 20      CMP #$20         Compare w/blank space ' ' value
CC81   F0 08      BEQ $CC8B        Identical?
CC83   C9 1D      CMP #$1D         NO—Test w/value for'Cursor right'
CC85   F0 04      BEQ $CC8B        Identical?
CC87   C9 2C      CMP #$2C         NO—Compare with comma value
CC89   D0 07      BNE $CC92        Identical?
CC8B3  C8         INY              YES—Buffer pointer to next char
CC8C   CC 74 02   CPY $0274        Test against command string value
CC8F   90 EB      BCC $CC7C        Pointer to end of input buffer?
CC91   60         RTS              YES—Return from this subroutine
CC921  20 A1 CC   JSR $CCA1        Get, compute and set parameters
```

```
CC95   EE 77 02   INC $0277       Current number of parameters
CC98   AC 79 02   LDY $0279       Total number of parameters
CC9B   E0 04      CPX #$04        Test with maximum # of parameters
CC9D   90 EC      BCC $CC8B       Too many parameters?
CC9F   B0 8A      BCS $CC2B       YES—Jump to $CC2B
-----------------------------------------------------------------------
[CC92]
Convert / set Block command parameters from ASCII to binary
CCA1   A9 00      LDA #$00        Clear range used
CCA3   85 6F      STA $6F         as temporary storage
CCA5   85 70      STA $70         for mathematical
CCA7   85 72      STA $72         operations
CCA9   A2 FF      LDX #$FF        Pointer to current math register
CCAB1  B9 00 02   LDA $0200,Y     Get next char from input buffer
CCAE   C9 40      CMP #$40        Compare with ASCII value for '@'
CCB0   B0 18      BCS $CCCA       Is there a character?
CCB2   C9 30      CMP #$30        NO—Test value for '0'
CCB4   90 14      BCC $CCCA       Is there a number?
CCB6   29 0F      AND #$0F        Compute numeric value and
CCB8   48         PHA             save it
CCB9   A5 70      LDA $70         Shift value in temp. storage
CCBB   85 71      STA $71         $6F-$71 range; move
CCBD   A5 6F      LDA $6F         one place to $71 so
CCBF   85 70      STA $70         that $6F will be free
CCC1   68         PLA             Repeat binary numbers and write
CCC2   85 6F      STA $6F         in temporary memory
CCC4   C8         INY             Buffer pointer to next character
CCC5   CC 74 02   CPY $0274       Check w/end position of params
CCC8   90 E1      BCC $CCAB       Entire decimal number read in?
CCCA2  8C 79 02   STY $0279       YES—Save current buffer pointer
CCCD   18         CLC             Initialize add routine
CCCE   A9 00      LDA #$00        'Dummy value' for first run of
CCD01  E8         INX             routine
CCD1   E0 03      CPX #$03        Test against max. decimal numbers
CCD3   B0 0F      BCS $CCE4       Are too many numbers given?
CCD5   B4 6F      LDY $6F,X       NO—Get value of a # in counter
CCD72  88         DEY             Decrement number
CCD8   30 F6      BMI $CCD0       Is decimal number zero?
CCDA   7D F2 CC   ADC $CCF2,X     NO—Get binary of number
CCDD   90 F8      BCC $CCD7       Add it; is binary number > 256 ?
CCDF   18         CLC             YES—Turn high-byte and
CCE0   E6 72      INC $72         increment by one
CCE2   D0 F3      BNE $CCD7       Jump to $CCD7
CCE41  48         PHA             Save equiv. binary value(lo-byte)
CCE5   AE 77 02   LDX $0277       Get parameter number
CCE8   A5 72      LDA $72         Enter binary value (high-byte)
```

```
CCEA  9D 80 02    STA $0280,X      in parameter table
CCED  68          PLA              Repeat lo-byte of binary value &
CCEE  9D 85 02    STA $0285,X      save it
CCF1  60          RTS              Return from this subroutine
---------------------------------------------------------------------
CCF2  01 0A 64                     Binary values for 1, 10 und 100
---------------------------------------------------------------------
[Origin at routine CC1B]
Block-Free command ('B-F'); Free block in BAM
CCF5  20 F5 CD    JSR $CDF5        Get track/sector number
CCF8  20 5F EF    JSR $EF5F        Set block bit to 'free'
CCFB  4C 94 C1    JMP $C194        Prepare acknowledgement and end
---------------------------------------------------------------------
CCFE  A9 01       LDA #$01         Unused program set from
CD00  8D F9 02    STA $02F9        CBM 4040 ROM
---------------------------------------------------------------------
[Origin at routine CC1B]
Block-Allocate command ('B-A')
CD03  20 F5 CD    JSR $CDF5        Get track/sector number
CD06  A5 81       LDA $81          Get sector number and
CD08  48          PHA              save it
CD09  20 FA F1    JSR $F1FA        Look for next free sector in BAM
CD0C  F0 0B       BEQ $CD19        Is block free?
CD0E  68          PLA              YES—Get number of desired sector;
CD0F  C5 81       CMP $81          Compare with current sector #
CD11  D0 19       BNE $CD2C        Identical?
CD13  20 90 EF    JSR $EF90        Identify BAM sector as allocated
CD16  4C 94 C1    JMP $C194        Prepare acknowledgement and end
CD19[1] 68        PLA              Adjust stack, clear sector number
CD1A[1] A9 00     LDA #$00         Newly establish
CD1C  85 81       STA $81          sector number
CD1E  E6 80       INC $80          Set track pointer to next track;
CD20  A5 80       LDA $80          get pointer
CD22  CD AC 02    CMP $02AC        Compare w/value of largst track+1
CD25  B0 0A       BCS $CD31        Is track number smaller?
CD27  20 FA F1    JSR $F1FA        YES—Look for next sector
CD2A  F0 EE       BEQ $CD1A        Found it?
CD2C[1] A9 65     LDA #$65         NO—Display
CD2E  20 45 E6    JSR $E645        '65 No Block' error
CD31[1] A9 65     LDA #$65         Display
CD33  20 C8 C1    JSR $C1C8        '65 No Block' error
---------------------------------------------------------------------
```

```
[CD42/CDA6]
Test 'B-R' parameters and read sector in buffer
CD36  20 F2 CD   JSR $CDF2       Test & get track / sector number
CD39  4C 60 D4   JMP $D460       Read sector in buffer
-------------------------------------------------------------------
[CD4A]
Get byte from buffer
CD3C  20 2F D1   JSR $D12F       Set buffer pointer
CD3F  A1 99      LDA ($99,X)     Get byte
CD41  60         RTS             Return from this subroutine
-------------------------------------------------------------------
[CD56/CD62]
Read sector from diskette to buffer; initialize pointer
CD42  20 36 CD   JSR $CD36       Get parameter and read sector
CD45  A9 00      LDA #$00        Determine position of buffr pntr
CD47  20 C8 D4   JSR $D4C8       Set buffer pointer
CD4A  20 3C CD   JSR $CD3C       Get a byte from buffer
CD4D  99 44 02   STA $0244,Y     Amount of data to be transferred
CD50  A9 89      LDA #$89        Free up channel for
CD52  99 F2 00   STA $00F2,Y     reading and writing
CD55  60         RTS             Return fom this subroutine
-------------------------------------------------------------------
[Origin at routine CC1B]
Routine for Block-Read command ('B-R'); Read sector from diskette
CD56  20 42 CD   JSR $CD42       Read sector and set pointer
CD59  20 EC D3   JSR $D3EC       Output byte from buffer
CD5C  4C 94 C1   JMP $C194       Prepare return message and end
-------------------------------------------------------------------
[Vector: FFEA]
Routine for U1-command (cf. B-R); read sector from diskette
CD5F  20 6F CC   JSR $CC6F       Get parameters
CD62  20 42 CD   JSR $CD42       Read sector in buffer
CD65  B9 44 02   LDA $0244,Y     Set # of bytes to be transferred
CD68  99 3E 02   STA $023E,Y     as bytes to be given out
CD6B  A9 FF      LDA #$FF        Re-initialize number of bytes
CD6D  99 44 02   STA $0244,Y     to be transferred
CD70  4C 94 C1   JMP $C194       Prepare return message and end
-------------------------------------------------------------------
[Origin at routine CC1B]
Routine for Block-Write command
CD73  20 F2 CD   JSR $CDF2       Allocate buffer and open channel
CD76  20 E8 D4   JSR $D4E8       Initialize and get buffer
CD79  A8         TAY             pointer
CD7A  88         DEY             Pointer to previous character
CD7B  C9 02      CMP #$02        Compare with start of data range
CD7D  B0 02      BCS $CD81       Is pointer correctly set?
```

```
CD7F   A0 01      LDY #$01        YES-Byte valu f/current buff pos.
CD81¹  A9 00      LDA #$00        Position of buffer pointer
CD83   20 C8 D4   JSR $D4C8       Get buffer pointer
CD86   98         TYA             Position in buffer
CD87   20 F1 CF   JSR $CFF1       write byte in buffer
CD8A   8A         TXA             Double and save
CD8B   48         PHA             buffer pointer
CD8C   20 64 D4   JSR $D464       Write sector to diskette
CD8F   68         PLA             Repeat buffer number and
CD90   AA         TAX             set it
CD91   20 AE FF   JSR $FFAE       Re-set buffer pointer
CD94   4C 94 C1   JMP $C194       Prepare return message and end
-----------------------------------------------------------------
[Vector: FFEC]
Routine for U2-command (cf.B-W); Write sector from buffer to disk
CD97   20 6F CC   JSR $CC6F       Get parameter from command string
CD9A   20 F2 CD   JSR $CDF2       Test and set parameter
CD9D   20 64 D4   JSR $D464       Write sector to disk
CDA0   4C 94 C1   JMP $C194       Prepare return message and end
-----------------------------------------------------------------
[Origin at routine CC1B]
Routine for Block-Execute-command ('B-E'); read sector and execute
CDA3   20 58 F2   JSR $F258       No function (rts)
CDA6   20 36 CD   JSR $CD36       Read sector in buffer
CDA9   A9 00      LDA #$00        Set buffer address (low-byte) to
CDAB   85 6F      STA $6F         start-of-buffer
CDAD   A6 F9      LDX $F9         Get buffer number
CDAF   BD E0 FE   LDA $FEE0,X     Get hi-byte of buffer address and
CDB2   85 70      STA $70         set in pointer at start-of-buffer
CDB4   20 BA CD   JSR $CDBA       Start program in buffer
CDB7   4C 94 C1   JMP $C194       Return at 'RTS'
CDBA¹  6C 6F 00   JMP ($006F)     Jump to pointer in buffer
-----------------------------------------------------------------
[Origin at routine CC1B]
Routine for Block-Pointer-command ('B-P'); set buffer pointer
CDBD   20 D2 CD   JSR $CDD2       Allocate buffer and open channel
CDC0   A5 F9      LDA $F9         Get buffer number
CDC2   0A         ASL A           double (buffer pointer as 2-Byte)
CDC3   AA         TAX             and save it
CDC4   AD 86 02   LDA $0286       Get new pos. of buffer pointer &
CDC7   95 99      STA $99,X       set as low-byte in buffer pointer
CDC9   20 2F D1   JSR $D12F       Get buffer and channel number
CDCC   20 EE D3   JSR $D3EE       Get byte frm current buffer pos.
CDCF   4C 94 C1   JMP $C194       Prepare return msg. and end
-----------------------------------------------------------------
```

```
[CDBD/CDF2]
Allocate buffer and open channel
CDD2  A6 D3     LDX $D3        Get parameter number
CDD4  E6 D3     INC $D3        Set to next assignment
CDD6  BD 85 02  LDA $0285,X    Get channel number from table
CDD9  A8        TAY            and save it
CDDA  88        DEY            Decrement channel number
CDDB  88        DEY            by 2 and compare with
CDDC  C0 0D     CPY #$0D       value for channel 14
CDDE  90 05     BCC $CDE5      Is the channel number < 15?
CDE0¹ A9 70     LDA #$70       NO—Display
CDE2  4C C8 C1  JMP $C1C8      '70 No Channel'
CDE5¹ 85 83     STA $83        Set channel # as 2ndary address
CDE7  20 EB D0  JSR $D0EB      and open channel
CDEA  B0 F4     BCS $CDE0      Channel already open?
CDEC  20 93 DF  JSR $DF93      NO—Get buffer number and
CDEF  85 F9     STA $F9        set it
CDF1  60        RTS            Return from this subroutine
-------------------------------------------------------------------
[CD03/CD36/CD73/CD9A]
Test paramters for valid sector assignment
CDF2  20 D2 CD  JSR $CDD2      Allocate buffer
CDF5  A6 D3     LDX $D3        Parameter number
CDF7  BD 85 02  LDA $0285,X    Get byte from temporary storage
CDFA  29 01     AND #$01       and isolate drive number; take on
CDFC  85 7F     STA $7F        as current drive
CDFE  BD 87 02  LDA $0287,X    Set number of desired
CE01  85 81     STA $81        track
CE03  BD 86 02  LDA $0286,X    Take on number of
CE06  85 80     STA $80        desired sector
CE08  20 5F D5  JSR $D55F      Test for valid track and sector
CE0B  4C 00 C1  JMP $C100      Switch on LED to current drive
-------------------------------------------------------------------
[E255/E338/E436]
Get record from relative file
CE0E  20 2C CE  JSR $CE2C      Determine # of bytes computed til
CE11  20 6E CE  JSR $CE6E      record and sector # of the record
CE14  A5 90     LDA $90        Get remainder of division &set as
CE16  85 D7     STA $D7        buffer pointer to start of record
CE18  20 71 CE  JSR $CE71      Get side-sector shown by record
CE1B  E6 D7     INC $D7        Adjust buffer pointer in physical
CE1D  E6 D7     INC $D7        sector to linked bytes
CE1F  A5 8B     LDA $8B        Get and save number of
CE21  85 D5     STA $D5        side-sector
CE23  A5 90     LDA $90        Get remainder of div. & calc.
CE25  0A        ASL A          position of sector pointer for
```

```
CE26   18          CLC               record in computed side-sector
CE27   69 10       ADC #$10          and
CE29   85 D6       STA $D6           save it
CE2B   60          RTS               Return from this subroutine
----------------------------------------------------------------
[CE0E]
Compute number of bytes up to record
CE2C   20 D9 CE    JSR $CED9         Clear temporary memory
CE2F   85 92       STA $92           Value in math register 2
CE31   A6 82       LDX $82           Get channel number (buffer) and
CE33   B5 B5       LDA $B5,X         determine and take on
CE35   85 90       STA $90           appropriate record # (low-byte)
CE37   B5 BB       LDA $BB,X         Get high-byte of record number &
CE39   85 91       STA $91           take it on
CE3B   D0 04       BNE $CE41         Record number greater than 255?
CE3D   A5 90       LDA $90           NO-Get low-byte of record number
CE3F   F0 0B       BEQ $CE4C         Is record number  = 0?
CE41[1] A5 90      LDA $90           NO-Get record number(low-byte)and
CE43   38          SEC               diminish by
CE44   E9 01       SBC #$01          one; take up new
CE46   85 90       STA $90           value
CE48   B0 02       BCS $CE4C         Is record number < 1?
CE4A   C6 91       DEC $91           YES-Then decrement hi-byte by one
CE4C[2] B5 C7      LDA $C7,X         Get record length and
CE4E   85 6F       STA $6F           save it
CE50[1] 46 6F      LSR $6F           Test against equal value
CE52   90 03       BCC $CE57         Is the record length the same?
CE54   20 ED CE    JSR $CEED         NO-Add reg. 2 to reg. 1
CE57[1] 20 E5 CE   JSR $CEE5         Math register times 2
CE5A   A5 6F       LDA $6F           Current record
CE5C   D0 F2       BNE $CE50         Compute bits
CE5E   A5 D4       LDA $D4           Pointer in position in Record
CE60   18          CLC               Count up current
CE61   65 8B       ADC $8B           math register by 1
CE63   85 8B       STA $8B           Re-set low-byte
CE65   90 06       BCC $CE6D         Has a transfer occurred?
CE67   E6 8C       INC $8C           YES-Adjust next byte
CE69   D0 02       BNE $CE6D         Another transfer occurred frm it?
CE6B   E6 8D       INC $8D           YES-Adjust highest byte
CE6D[2] 60         RTS               Return from this subroutine
----------------------------------------------------------------
[CE11]
Division of math register by 254 (sector length)
CE6E   A9 FE       LDA #$FE          Set value of divisor (254)
CE70   2C          .byte $2C         Jump two bytes (bit command)
----------------------------------------------------------------
```

```
[CE18]
Division of math register by 120 (record entries in side-sector)
CE71   A9 78       LDA #$78        Set value of divisor (120)
CE73   85 6F       STA $6F         and save it
CE75   A2 03       LDX #$03        Number of bytes per math register
CE77¹  B5 8F       LDA $8F,X       Recover current
CE79   48          PHA             contents
CE7A   B5 8A       LDA $8A,X       Copy range $88-$8A to
CE7C   95 8F       STA $8F,X       register 2
CE7E   68          PLA             Contents of previous reg. 2 in
CE7F   95 8A       STA $8A,X       range $88-$8A (exchange)
CE81   CA          DEX             Pointer to next byte
CE82   D0 F3       BNE $CE77       Entire register exchanged?
CE84   20 D9 CE    JSR $CED9       YES—Clear register 1
CE87¹  A2 00       LDX #$00        Initialize counter
CE89¹  B5 90       LDA $90,X       Get byte from reg. 2 & prepare
CE8B   95 8F       STA $8F,X       for shifting by one byte
CE8D   E8          INX             Pointer to next byte
CE8E   E0 04       CPX #$04        Compare with # of register bytes
CE90   90 F7       BCC $CE89       Entire register shifted?
CE92   A9 00       LDA #$00        YES—Clear most significant
CE94   85 92       STA $92         byte
CE96   24 6F       BIT $6F         Test divisor
CE98   30 09       BMI $CEA3       Is it greater than 128?
CE9A   06 8F       ASL $8F         NO—Put bit0 frm least signif.part
CE9C   08          PHP             of reg. 2 in carry and save it
CE9D   46 8F       LSR $8F         Re-establish register
CE9F   28          PLP             Repeat carry and
CEA0   20 E6 CE    JSR $CEE6       shift in reg. 2
CEA3¹  20 ED CE    JSR $CEED       Add register 1 to register 2
CEA6   20 E5 CE    JSR $CEE5       Double register 2
CEA9   24 6F       BIT $6F         Test divisor
CEAB   30 03       BMI $CEB0       Is it greater than 128?
CEAD   20 E2 CE    JSR $CEE2       NO—Take register 24 times
CEB0¹  A5 8F       LDA $8F         Add to previous
CEB2   18          CLC             value in
CEB3   65 90       ADC $90         reg. 2 and
CEB5   85 90       STA $90         save result down
CEB7   90 06       BCC $CEBF       Has a transfer occurred?
CEB9   E6 91       INC $91         YES—Adjust 2nd byte of register
CEBB   D0 02       BNE $CEBF       Transfer also a result of this?
CEBD   E6 92       INC $92         YES—Set highest byte of
CEBF²  A5 92       LDA $92         register
CEC1   05 91       ORA $91         Combine 2nd byte
CEC3   D0 C2       BNE $CE87       Both bytes 0(register value<256)?
CEC5   A5 90       LDA $90         YES—Get least signif. reg. byte
```

```
CEC7   38         SEC                 and pull a divisor
CEC8   E5 6F      SBC $6F             from that
CECA   90 0C      BCC $CED8           Transfer occurred?
CECC   E6 8B      INC $8B             NO—Increment register 1
CECE   D0 06      BNE $CED6           Transfer?
CED0   E6 8C      INC $8C             Adjust 2nd byte
CED2   D0 02      BNE $CED6           Transfer?
CED4   E6 8D      INC $8D             Adjust last byte
CED6²  85 90      STA $90             Set new value
CED8¹  60         RTS                 Return from this subroutine
------------------------------------------------------------------
[CE2C/CE84]
Clear math register 1 ($8B/$8C/$8D)
CED9   A9 00      LDA #$00            Value which should be
CEDB   85 8B      STA $8B             cleared in
CEDD   85 8C      STA $8C             math register
CEDF   85 8D      STA $8D             when transferred
CEE1   60         RTS                 Return from this subroutine
------------------------------------------------------------------
[CEAD]
Multiply math register 2 ($90/$91/$92) four times
CEE2   20 E5 CE   JSR $CEE5           Double register contents
------------------------------------------------------------------
[CE57/CEA6/CEE2/CEE6:CEA0]
Double math register 2 ($90/$91/$92)
CEE5   18         CLC                 Value to be shifted = 0
CEE6¹  26 90      ROL $90             Shift value in register and
CEE8   26 91      ROL $91             shift entire register by
CEEA   26 92      ROL $92             one bitposition to the left
CEEC   60         RTS                 Return from this subroutine
------------------------------------------------------------------
[CE54/CEA3]
Add math register 2 ($90/$91/$92) to math register 1 ($8B/$8C/$8D)
CEED   18         CLC                 Begin addition
CEEE   A2 FD      LDX #$FD            # of bytes in registr(neg. value)
CEF0¹  B5 8E      LDA $8E,X           Get byte from register 1
CEF2   75 93      ADC $93,X           Get value from register 2, & add;
CEF4   95 8E      STA $8E,X           store result in register 1
CEF6   E8         INX                 Set pointer to next number
CEF7   D0 F7      BNE $CEF0           Entire register added?
CEF9   60         RTS                 YES—Return from this subroutine
------------------------------------------------------------------
```

```
[CF17/EBBF]
Initialize buffer channel table
CEFA   A2 00      LDX #$00       Start of buffer 0
CEFC1  8A         TXA            Channel assignment number (0)
CEFD   95 FA      STA $FA,X      Clear channel assignmnt of buffer
CEFF   E8         INX            Choose next buffer
CF00   E0 04      CPX #$04       Test against highest-#ed buffer
CF02   D0 F8      BNE $CEFC      All buffers already worked on?
CF04   A9 06      LDA #$06       YES—Use buffer 4 for
CF06   95 FA      STA $FA,X      channel 6 (BAM)
CF08   60         RTS            Return from this subroutine
---------------------------------------------------------------
[CF1E/CF7B]
Test channel number in buffer channel table
CF09   A0 04      LDY #$04       Number of buffers
CF0B   A6 82      LDX $82        Number of channels sought
CF0D1  B9 FA 00   LDA $00FA,Y    Pre-arranged channel # of buffer
CF10   96 FA      STX $FA,Y      Set new number
CF12   C5 82      CMP $82        Compare old number with new
CF14   F0 07      BEQ $CF1D      Both equal?
CF16   88         DEY            NO—Go to next buffer
CF17   30 E1      BMI $CEFA      Was that the last buffer?
CF19   AA         TAX            NO—Take on old channel number
CF1A   4C 0D CF   JMP $CF0D      and test it
CF1D1  60         RTS            Return from this subroutine
---------------------------------------------------------------
[BF5A/D0B7/D0C0/D16A/D180/D18C/D1BB/DB2F/DB7D/DBA2/E04A/E05D/E072/E078]
[E18D/E19A/E19D/E2B9/E3B6/E3C8/E439/E451]
Manage and assign buffer
CF1E   20 09 CF   JSR $CF09      Actualize buffer table
CF21   20 B7 DF   JSR $DFB7      Get status of chosen buffer
CF24   D0 46      BNE $CF6C      Is buffer free?
CF26   20 D3 D1   JSR $D1D3      YES—Set buffer of appropriate drv
CF29   20 8E D2   JSR $D28E      Look for buffer
CF2C   30 48      BMI $CF76      Buffer been found?
CF2E   20 C2 DF   JSR $DFC2      YES—Activate buffer
CF31   A5 80      LDA $80        Save current track
CF33   48         PHA            number
CF34   A5 81      LDA $81        Save current sector
CF36   48         PHA            number
CF37   A9 01      LDA #$01       Pointer to position in buffer
CF39   20 F6 D4   JSR $D4F6      Get a byte from buffer and save
CF3C   85 81      STA $81        as sector number
CF3E   A9 00      LDA #$00       Pointer to position in buffer
CF40   20 F6 D4   JSR $D4F6      Get byte from buffer and save
CF43   85 80      STA $80        as track number
```

```
CF45   F0 1F       BEQ $CF66       Any more sectors in string?
CF47   20 25 D1    JSR $D125       YES—at current filetype
CF4A   F0 0B       BEQ $CF57       Sector belong to a REL file?
CF4C   20 AB DD    JSR $DDAB       NO—Test last jobcode
CF4F   D0 06       BNE $CF57       Was it a write procedure?
CF51   20 8C CF    JSR $CF8C       YES—Change buffer status (in/out)
CF54   4C 5D CF    JMP $CF5D       and continue
CF57²  20 8C CF    JSR $CF8C       Change buff stat(active/passive)
CF5A   20 57 DE    JSR $DE57       Set 'Read sector' jobcode
CF5D¹  68          PLA             Re-establish current
CF5E   85 81       STA $81         sector number
CF60   68          PLA             Re-establish current
CF61   85 80       STA $80         track number;
CF63   4C 6F CF    JMP $CF6F       continue
CF66¹  68          PLA             Re-establish current
CF67   85 81       STA $81         sector number
CF69   68          PLA             Re-establish current
CF6A   85 80       STA $80         track number
CF6C¹  20 8C CF    JSR $CF8C       Change buff stat(active/passive)
CF6F¹  20 93 DF    JSR $DF93       Get buffer number and
CF72   AA          TAX             save it;
CF73   4C 99 D5    JMP $D599       wait until job is executed
CF76²  A9 70       LDA #$70        Display
CF78   4C C8 C1    JMP $C1C8       '70 No Channel' error message
-------------------------------------------------------------------
[E325]
Look for free buffer
CF7B   20 09 CF    JSR $CF09       Actualize buffer table
CF7E   20 B7 DF    JSR $DFB7       Get number of a buffer
CF81   D0 08       BNE $CF8B       Is buffer free?
CF83   20 8E D2    JSR $D28E       NO—Choose another buffer
CF86   30 EE       BMI $CF76       Has another buffer been found?
CF88   20 C2 DF    JSR $DFC2       YES—Activate buffer
CF8B¹  60          RTS             Return from this subroutine
-------------------------------------------------------------------
[CF51/CF57/CF6C]
Toggle buffer from active to passive and back
CF8C   A6 82       LDX $82         Current channel number
CF8E   B5 A7       LDA $A7,X       Get corresponding buffer status
CF90   49 80       EOR #$80        Change flag for buffer in/out and
CF92   95 A7       STA $A7,X       write it back in
CF94   B5 AE       LDA $AE,X       Get number of 2nd buffer and
CF96   49 80       EOR #$80        switch over
CF98   95 AE       STA $AE,X       Write new value in table
CF9A   60          RTS             Return from this subroutine
-------------------------------------------------------------------
```

```
[C9D5/C9E7]
Write bytes over internal channel in buffer
CF9B  A2 12     LDX #$12     Set number of write channel (18)
CF9D  86 83     STX $83      as current secondary address
CF9F  20 07 D1  JSR $D107    Look for channel and open
CFA2  20 00 C1  JSR $C100    Current drive's LED on
CFA5  20 25 D1  JSR $D125    Get corresponding filetype
CFA8  90 05     BCC $CFAF    Relative file?
CFAA  A9 20     LDA #$20     YES—Clear 'File not closed'
CFAC  20 9D DD  JSR $DD9D    flag
CFAF1 A5 83     LDA $83      Get current secondary address
CFB1  C9 0F     CMP #$0F     Compare with command channel #
CFB3  F0 23     BEQ $CFD8    Is command channel required?
CFB5  D0 08     BNE $CFBF    NO—Jump to $CBBF
--------------------------------------------------------------
```

```
[835C/EA48]
Write byte into file
CFB7  A5 84     LDA $84      Last secondary address
CFB9  29 8F     AND #$8F     Get channel number and test
CFBB  C9 0F     CMP #$0F     against command channel
CFBD  B0 19     BCS $CFD8    Has file channel been chosen?
CFBF  20 25 D1  JSR $D125    Get current filetype
CFC2  B0 05     BCS $CFC9    'REL' or 'USR'?
CFC4  A5 85     LDA $85      NO—Get current file byte & write
CFC6  4C 9D D1  JMP $D19D    in current buffer
CFC91 D0 03     BNE $CFCE    Is type a relative file?
CFCB  4C AB E0  JMP $E0AB    YES—Take byte in current record
CFCE1 A5 85     LDA $85      Get current filebyte and write
CFD0  20 F1 CF  JSR $CFF1    in buffer
CFD3  A4 82     LDY $82      Get number of current channel
CFD5  4C EE D3  JMP $D3EE    Get next byte for output
CFD82 A9 04     LDA #$04     Get highest channel number (4)
CFDA  85 82     STA $82      as command channel number
CFDC  20 E8 D4  JSR $D4E8    Initialize buffer pointer;
CFDF  C9 2A     CMP #$2A     test for end-of-buffer
CFE1  F0 05     BEQ $CFE8    Is buffer full?
CFE3  A5 85     LDA $85      NO—Get current data byte and put
CFE5  20 F1 CF  JSR $CFF1    in buffer
CFE81 A5 F8     LDA $F8      Test flag for last byte (EOI)
CFEA  F0 01     BEQ $CFED    No more data?
CFEC  60        RTS          YES—Return from this subroutine
CFED1 EE 55 02  INC $0255    Clear command mode flag
CFF0  60        RTS          Return from this subroutine
--------------------------------------------------------------
```

```
[CD87/CFD0/CFE5/D19D/D1B0/D1B5/D4A8/D4AD/D4BB/D4C0/D4C5/D74D/D754/D75B]
[DB73/DB95/DB99/ECBE/ECC3/ECC8/ECCB/ECD1/ECD6/ECE7/ECEC/ECEF/ECFA/ED00]
[ED08/ED26/ED2C/ED3D/ED40/ED43/ED5E/CFFD:DD92]
Write byte in current buffer
CFF1   48         PHA             Save byte
CFF2   20 93 DF   JSR $DF93       Get number of buffer
CFF5   10 06      BPL $CFFD       Is buffer properly set up?
CFF7   68         PLA             NO—Correct stack
CFF8   A9 61      LDA #$61        Display
CFFA   4C C8 C1   JMP $C1C8       '61 File Not Open' message
CFFD2  0A         ASL A           Double buffer number and
CFFE   AA         TAX             save it
CFFF   68         PLA             Repeat byte and write
D000   81 99      STA ($99,X)     in current buffer
D002   F6 99      INC $99,X       Set buffer pointer to next char
D004   60         RTS             Return from this subroutine
---------------------------------------------------------------
[Origin at C146]
Initialize command routine ('i')
D005   20 D1 C1   JSR $C1D1       Get parameters
D008   20 42 D0   JSR $D042       Read BAM from diskette
D00B   4C 94 C1   JMP $C194       Prepare return message and end
---------------------------------------------------------------
[C64C/D048]
Initialize current drive
D00E   20 0F F1   JSR $F10F       Get channel number and
D011   A8         TAY             save it
D012   B6 A7      LDX $A7,Y       Get corresponding buffer status
D014   E0 FF      CPX #$FF        Compare with 'occupied' flag
D016   D0 14      BNE $D02C       Is buffer free?
D018   48         PHA             YES—Save channel number
D019   20 8E D2   JSR $D28E       Look for buffer and set pointer
D01C   AA         TAX             Get buffer number
D01D   10 05      BPL $D024       Buffer found?
D01F   A9 70      LDA #$70        NO—display
D021   20 48 E6   JSR $E648       '70 No Channel' message
D0241  68         PLA             Repeat channel number and
D025   A8         TAY             save it
D026   8A         TXA             Get buffer number
D027   09 80      ORA #$80        Flag value for buffer active
D029   99 A7 00   STA $00A7,Y     Write to channel buffer table
D02C1  8A         TXA             Get buffer number and
D02D   29 0F      AND #$0F        set flags out
D02F   85 F9      STA $F9         Save current buffer number
D031   A2 00      LDX #$00        Set current sector
D033   86 81      STX $81         number
```

```
D035  AE 85 FE   LDX $FE85        Set number of directory track as
D038  86 80      STX $80          current track number
D03A  20 D3 D6   JSR $D6D3        Set track/sector for jobloop
D03D  A9 B0      LDA #$B0         Jobcode for 'Search sector'
D03F  4C E5 A6   JMP $A5C5        Initialize diskette
-------------------------------------------------------------
[8FE1/907C/C666/D008/D828/E63E/ED87/EE46/EEB1]
Read BAM in buffer
D042  20 D1 F0   JSR $F0D1        Clear track number for BAM
D045  20 13 D3   JSR $D313        Close other drive channel
D048  20 0E D0   JSR $D00E        Initialize drive
D04B  A6 7F      LDX $7F          Get current drive number and set
D04D  A9 00      LDA #$00         appropriate flag for
D04F  9D 51 02   STA $0251,X      'Valid BAM'
D052  8A         TXA              Double drive
D053  0A         ASL A            (number  for 2-drive pointer)
D054  AA         TAX              and save it
D055  A5 16      LDA $16          Get/save first blockheader ID
D057  95 12      STA $12,X        character
D059  A5 17      LDA $17          Get/save second blockheader
D05B  95 13      STA $13,X        ID character; take it all up
D05D  20 67 A6   JSR $A667        Read BAM from diskette
D060  A5 F9      LDA $F9          Get number of current buffer
D062  0A         ASL A            and double it
D063  AA         TAX              (address held in 2 bytes)
D064  A9 02      LDA #$02         Arrange lo-byte of buffer address
D066  95 99      STA $99,X        in buffer table
D068  A1 99      LDA ($99,X)      Get byte from buffer
D06A  A6 7F      LDX $7F          Get current drive number
D06C  9D 01 01   STA $0101,X      Store byte as format identifier
D06F  A9 00      LDA #$00         Clear disk exchange flag &pre-set
D071  4C 1D AA   JMP $AA1D        'Drive ready' flag
D074  EA         NOP              Unused byte
-------------------------------------------------------------
[A83B/AA22/EEF1]
Compute total number of blocks free
D075  20 3A EF   JSR $EF3A        set buffer addr in pnters $6D/$6E
D078  A0 04      LDY #$04         Set buffer pntr to begin.of BAM
D07A  A9 00      LDA #$00         Initialize
D07C  AA         TAX              block counter
D07D1 18         CLC              Get # of free track blocks from
D07E  71 6D      ADC ($6D),Y      BAM and add to counter
D080  90 01      BCC $D083        Has a transfer occurred?
D082  E8         INX              YES—Increment hi-byte of pointer
```

```
D083² C8         INY            Set buffer pointer to # of blocks
D084  C8         INY            free to the
D085  C8         INY            next track;
D086  C8         INY            jump to sector bitpattern
D087  C0 48      CPY #$48       Test pntr against pos.of track18
D089  F0 F8      BEQ $D083      Pointer points to valu f/track18?
D08B  C0 90      CPY #$90       NO—Test for last track
D08D  D0 EE      BNE $D07D      Add free blocks to all tracks?
D08F  48         PHA            YES—Save block counter (low-byte)
D090  8A         TXA            Get high-byte of block counter
D091  A6 7F      LDX $7F        Get drive # and save free blocks
D093  9D FC 02   STA $02FC,X    on drive
D096  68         PLA            Get low-byte of free blocks
D097  4C 51 A9   JMP $A951      Compute number of 1571 blocks
D09A  60         RTS            Return from this subroutine
----------------------------------------------------------------
[D0AF/DC57]
Read sector from diskette to buffer
D09B  20 D0 D6   JSR $D6D0      Track/sector number to jobloop
D09E  20 C3 D0   JSR $D0C3      Give jobcode for 'Read sector'
D0A1  20 99 D5   JSR $D599      Wait til sector  read into buffer
D0A4  20 37 D1   JSR $D137      Get 1st byte from buffer &save as
D0A7  85 80      STA $80        track of next sector
D0A9  20 37 D1   JSR $D137      Get next byte from buffer,set as
D0AC  85 81      STA $81        sector number of next sector
D0AE  60         RTS            Return from this subroutine
----------------------------------------------------------------
[E2CD]
Read in given sector and sector after that
D0AF  20 9B D0   JSR $D09B      Read sector from diskette
D0B2  A5 80      LDA $80        Get track
D0B4  D0 01      BNE $D0B7      Any more sectors onhand?
D0B6  60         RTS            NO—Return from this subroutine
D0B7¹ 20 1E CF   JSR $CF1E      Lay out another buffer and
D0BA  20 D0 D6   JSR $D6D0      parameters of next sector
D0BD  20 C3 D0   JSR $D0C3      Also read into next buffer
D0C0  4C 1E CF   JMP $CF1E      Re-activate first buffer
----------------------------------------------------------------
[D09E/D0BD/D189]
Read sector from diskette
D0C3  A9 80      LDA #$80       Set up jobcode for 'Read sector'
D0C5  D0 02      BNE $D0C9      Jump to $D0C9
----------------------------------------------------------------
```

```
[D1B8/D4B0/DB9C]
Write sector to diskette
D0C7   A9 90      LDA #$90     Set 'Write sector'
D0C9¹  8D 4D 02   STA $024D    Save jobcode
D0CC   20 93 DF   JSR $DF93    Get and save current buffer
D0CF   AA         TAX          number
D0D0   20 06 D5   JSR $D506    Test track/sector numbers
D0D3   8A         TXA          Repeat buffer number and
D0D4   48         PHA          hold onto it
D0D5   0A         ASL A        Double number
D0D6   AA         TAX          (2-byte values)
D0D7   A9 00      LDA #$00     Set back buffer address
D0D9   95 99      STA $99,X    (low-byte)
D0DB   20 25 D1   JSR $D125    Get current filetype
D0DE   C9 04      CMP #$04     Test for SEQ file identifier
D0E0   B0 06      BCS $D0E8    Sector belong to a SEQ file?
D0E2   F6 B5      INC $B5,X    Number of laid-out file blocks +1
D0E4   D0 02      BNE $D0E8    Has a transfer occurred?
D0E6   F6 BB      INC $BB,X    YES—High-byte of block pointer +1
D0E8²  68         PLA          Repeat buffer number and
D0E9   AA         TAX          set it
D0EA   60         RTS          Return from this subroutine
-------------------------------------------------------------
[81EB/C6E2/C9C3/CB45/CDE7/D39B/D90E/DE36/E20F/E680/E90A]
Open channel for reading
D0EB   A5 83      LDA $83      Get curent secondary address and
D0ED   C9 13      CMP #$13     compare w/maximum 2ndary address
D0EF   90 02      BCC $D0F3    Is address in allowable range?
D0F1   29 0F      AND #$0F     Limit 2ndary addresses to 0-15 &
D0F3¹  C9 0F      CMP #$0F     test for channel 15
D0F5   D0 02      BNE $D0F9    Is command channel communicated?
D0F7   A9 10      LDA #$10     YES—Set secondary address to 16
D0F9¹  AA         TAX          (error channel)
D0FA   38         SEC          Flag:'Channel not for reading'
D0FB   BD 2B 02   LDA $022B,X  Test channel status
D0FE   30 06      BMI $D106    Is channel in read mode?
D100   29 0F      AND #$0F     YES--Establish internal channel #
D102   85 82      STA $82      Set and save as current
D104   AA         TAX          channel number
D105   18         CLC          Set 'channel open' flag
D106¹  60         RTS          Return from this subroutine
-------------------------------------------------------------
```

```
[8343/CA63/CF9F/DB1B/DC43/E688/EA2F]
Search for and open channel
D107   A5 83      LDA $83       Get current secondary address and
D109   C9 13      CMP #$13      compare with maximum value (19)
D10B   90 02      BCC $D10F     Is address in allowable range?
D10D   29 0F      AND #$0F      NO—then convert and save
D10F1  AA         TAX           range
D110   BD 2B 02   LDA $022B,X   Get corresponding channel status
D113   A8         TAY           and save
D114   0A         ASL A         Turn status to test bit 6/7
D115   90 0A      BCC $D121     Is flag set for writing?
D117   30 0A      BMI $D123     YES—is flag set for reading?
D1191  98         TYA           NO—get channel status again
D11A   29 0F      AND #$0F      Establish pure channel number
D11C   85 82      STA $82       Set as current channel
D11E   AA         TAX           and save it
D11F   18         CLC           Set 'channel open' flag
D120   60         RTS           Return from this subroutine
D1211  30 F6      BMI $D119     Is flag for read set?
D1231  38         SEC           YES—set flag:'channel read only'
D124   60         RTS           Return from this subroutine
------------------------------------------------------------------
[C979/C9C6/C9E2/CA2B/CA48/CF47/CFA5/CFBF/D0DB/D3AC/D3C0/DB10/DBDE]
[E21E/E68E]
Get current file type
D125   A6 82      LDX $82       Get current channel number and
D127   B5 EC      LDA $EC,X     and get appropriate filetype
D129   4A         LSR A         Ignore drive number
D12A   29 07      AND #$07      Establish identifier for filetype
D12C   C9 04      CMP #$04      and compare w/ REL file code
D12E   60         RTS           Return from this subroutine
------------------------------------------------------------------
[CD3C/CDC9/D137/D3DE/E01D/E127/E138/E156]
Get channel and matching buffer number
D12F   20 93 DF   JSR $DF93     Get current buffer number and
D132   0A         ASL A         double it
D133   AA         TAX           Save as 2-byte value in pointer
D134   A4 82      LDY $82       Store as current channel number
D136   60         RTS           Return from this subroutine
------------------------------------------------------------------
[D0A4/D0A9/D156/D172/D17B/D192/D433/DAAA/DE9A/DE9F/ED67/EDF3/EDF8]
Get byte from current buffer
D137   20 2F D1   JSR $D12F     Set channel and buffer number
D13A   B9 44 02   LDA $0244,Y   pointer to end of buffer
D13D   F0 12      BEQ $D151     Is last byte of buffer read?
D13F   A1 99      LDA ($99,X)   NO—get byte from buffer
```

```
D141   48         PHA          and save
D142   B5 99      LDA $99,X    Get buffer pointer (low-byte)
D144   D9 44 02   CMP $0244,Y  & check against logical buff. end
D147   D0 04      BNE $D14D    End of the buffer reached?
D149   A9 FF      LDA #$FF     YES-set buffer pointer to the
D14B   95 99      STA $99,X    physical end-of-buffer
D14D1  68         PLA          and get another data byte
D14E   F6 99      INC $99,X    Set buff pnter to start-of-buffer
D150   60         RTS          Return from this subroutine
D1511  A1 99      LDA ($99,X)  Get last byte of buffer and reset
D153   F6 99      INC $99,X    buffer pointer to beginning
D155   60         RTS          Return from this subroutine
-----------------------------------------------------------------
[C899/C89E/D400/D45C/DCA9]
Get byte from file
D156   20 37 D1   JSR $D137    Get byte from buffer
D159   D0 36      BNE $D191    Was that the last byte in buffer?
D15B   85 85      STA $85      YES-Save data byte
D15D   B9 44 02   LDA $0244,Y  Get pointer f/correct buffr range
D160   F0 08      BEQ $D16A    Reached the physical end?
D162   A9 80      LDA #$80     NO-set flag in channel
D164   99 F2 00   STA $00F2,Y  status table for 'read'
D167   A5 85      LDA $85      Get another data byte
D169   60         RTS          Return from this subroutine
D16A1  20 1E CF   JSR $CF1E    Read next logical sector
D16D   A9 00      LDA #$00     Reset
D16F   20 C8 D4   JSR $D4C8    Buffer pointer
D172   20 37 D1   JSR $D137    Get 1st byte from sector and test
D175   C9 00      CMP #$00     against 'last sector' identifier
D177   F0 19      BEQ $D192    No more sectors on hand?
D179   85 80      STA $80      NO-Track number of next sector
D17B   20 37 D1   JSR $D137    Get second byte from sector
D17E   85 81      STA $81      and store as sector number
D180   20 1E CF   JSR $CF1E    Still a buffer laid out
D183   20 D3 D1   JSR $D1D3    Set buffer and drive number
D186   20 D0 D6   JSR $D6D0    Track & sector number on jobloop
D189   20 C3 D0   JSR $D0C3    Read sector to buffer
D18C   20 1E CF   JSR $CF1E    Switch back to previous buffer
D18F   A5 85      LDA $85      Get another data byte
D1911  60         RTS          Return from this subroutine
D1921  20 37 D1   JSR $D137    Get byte from buffer
D195   A4 82      LDY $82      Get current channel number
D197   99 44 02   STA $0244,Y  Set # of bytes to be transferred
D19A   A5 85      LDA $85      Get data byte again
D19C   60         RTS          Return from this subroutine
-----------------------------------------------------------------
```

```
[CFC6/D1A3:DA3D]
Write byte in file
D19D  20 F1 CF   JSR $CFF1      Write data byte in buffer
D1A0  F0 01      BEQ $D1A3      Is buffer full yet?
D1A2  60         RTS            NO—return from this subroutine
D1A3² 20 D3 D1   JSR $D1D3      Set buffer and drive number
D1A6  20 1E F1   JSR $F11E      Get next free sector from BAM
D1A9  A9 00      LDA #$00       Set buffer pointer on
D1AB  20 C8 D4   JSR $D4C8      string bytes of sector
D1AE  A5 80      LDA $80        Write track # of next sector in
D1B0  20 F1 CF   JSR $CFF1      string bytes of sector
D1B3  A5 81      LDA $81        Write next sector number into
D1B5  20 F1 CF   JSR $CFF1      sector string bytes
D1B8  20 C7 D0   JSR $D0C7      Write sector to diskette
D1BB  20 1E CF   JSR $CF1E      Change to next buffer
D1BE  20 D0 D6   JSR $D6D0      Set track and sector # for job
D1C1  A9 02      LDA #$02       Set buffer pointer to start of
D1C3  4C C8 D4   JMP $D4C8      data range
--------------------------------------------------------------------
[C623]
Set current buffer pointer to next character
D1C6  85 6F      STA $6F        Save new pointer position
D1C8  20 E8 D4   JSR $D4E8      Set pointer to current buffer and
D1CB  18         CLC            add to the new
D1CC  65 6F      ADC $6F        pointer value
D1CE  95 99      STA $99,X      Put new value in pointer lo-byte
D1D0  85 94      STA $94        and directory buffer pointer
D1D2  60         RTS            Return from this subroutine
--------------------------------------------------------------------
[CA53/CA66/CF26/D183/D1A3/E03C/E31C]
Get number of drive-assigned buffer
D1D3  20 93 DF   JSR $DF93      Determine and save buffer
D1D6  AA         TAX            number
D1D7  BD 5B 02   LDA $025B,X    Get coresponding jobcode frm tbl
D1DA  29 01      AND #$01       and from it compute drive number;
D1DC  85 7F      STA $7F        store as current drive
D1DE  60         RTS            Return from this subroutine
--------------------------------------------------------------------
[DCDF]
Look for write channel and buffer
D1DF  38         SEC            Set write flag
D1E0  B0 01      BCS $D1E3      Jump to $D1E3
--------------------------------------------------------------------
```

```
[91D0/CB9A/CBDF/DC48/ECA4/D20F:DC7E,DD13]
Look for read channel and buffer
D1E2  18        CLC          Set read flag
D1E3¹ 08        PHP          Save flag
D1E4  85 6F     STA $6F      Number of buffer being sought
D1E6  20 27 D2  JSR $D227    Clear all channels
D1E9  20 7F D3  JSR $D37F    Seek & lay out next free channel
D1EC  85 82     STA $82      Save channel number
D1EE  A6 83     LDX $83      Get secondary address
D1F0  28        PLP          Get read/write flag again
D1F1  90 02     BCC $D1F5    Should a read channel be opened?
D1F3  09 80     ORA #$80     NO—set 'write' flag and write
D1F5¹ 9D 2B 02  STA $022B,X  to status table
D1F8  29 3F     AND #$3F     Establish and save number of
D1FA  A8        TAY          internal channels
D1FB  A9 FF     LDA #$FF     Appropriate buffers
D1FD  99 A7 00  STA $00A7,Y  one and two
D200  99 AE 00  STA $00AE,Y  freed up
D203  99 CD 00  STA $00CD,Y  Third buffer freed up
D206  C6 6F     DEC $6F      Decrement # of buffer sought
D208  30 1C     BMI $D226    Found enough buffers?
D20A  20 8E D2  JSR $D28E    NO—look for a free buffer
D20D  10 08     BPL $D217    Find a buffer?
D20F³ 20 5A D2  JSR $D25A    NO—free up a buffer
D212  A9 70     LDA #$70     Error message
D214  4C C8 C1  JMP $C1C8    '70 No Channel' displayed
D217¹ 99 A7 00  STA $00A7,Y  Buffer number in map table
D21A  C6 6F     DEC $6F      Decrement # of buffers sought
D21C  30 08     BMI $D226    Found enough buffers?
D21E  20 8E D2  JSR $D28E    NO—look for next buffer
D221  30 EC     BMI $D20F    Found a free buffer?
D223  99 AE 00  STA $00AE,Y  YES—Save buffer number
D226² 60        RTS          Return from this subroutine
-----------------------------------------------------------------
[C8AA/D1E6/D30B/D331/D4DE/D4E5/DACE/DB29/DB5F/E695/EE01]
Free up channel
D227  A5 83     LDA $83      Get current 2ndary address and
D229  C9 0F     CMP #$0F     compare w/value f/command channel
D22B  D0 01     BNE $D22E    Is channel 15 active?
D22D  60        RTS          YES—return from this subroutine
D22E¹ A6 83     LDX $83      Get current secondary adddress
D230  BD 2B 02  LDA $022B,X  Determine proper channel status &
D233  C9 FF     CMP #$FF     test against'channel unused'value
D235  F0 22     BEQ $D259    Is channel free?
D237  29 3F     AND #$3F     NO—calculate channel number
D239  85 82     STA $82      and save it
```

```
D23B   A9 FF       LDA #$FF        Store flag value:'channel &
D23D   9D 2B 02    STA $022B,X     buffer free' in channel table
D240   A6 82       LDX $82         Get current channel number again
D242   A9 00       LDA #$00        Clear channel status
D244   95 F2       STA $F2,X       flags in channel table
D246   20 5A D2    JSR $D25A       Free up appropriate buffer
D249   A6 82       LDX $82         Current channel number
D24B   A9 01       LDA #$01        Bitflag for 'channel free'
D24D1  CA          DEX             Decrement channel number
D24E   30 03       BMI $D253       Is flag in correct position?
D250   0A          ASL A           NO—Give bitflag in bit pattern
D251   D0 FA       BNE $D24D       Jump to $D24D
D2531  0D 56 02    ORA $0256       Write flag in bit list of
D256   8D 56 02    STA $0256       the laid-out channel
D2591  60          RTS             Return from this subroutine
-----------------------------------------------------------------
[D20F/D246]
Free up buffer and corresponding channel
D25A   A6 82       LDX $82         Get current channel and
D25C   B5 A7       LDA $A7,X       determine buffer number for same
D25E   C9 FF       CMP #$FF        Compare with 'buffer free'
D260   F0 09       BEQ $D26B       Is buffer assigned that channel?
D262   48          PHA             YES—save buffer number and
D263   A9 FF       LDA #$FF        free up buffer
D265   95 A7       STA $A7,X       buffer table
D267   68          PLA             Get buffer number again
D268   20 F3 D2    JSR $D2F3       Free up bufer layout
D26B1  A6 82       LDX $82         Number of current channel
D26D   B5 AE       LDA $AE,X       Get corresponding buffer # and
D26F   C9 FF       CMP #$FF        test against 'not occupied' value
D271   F0 09       BEQ $D27C       Is the buffer free?
D273   48          PHA             NO—Save buffer number
D274   A9 FF       LDA #$FF        Free up buffer of
D276   95 AE       STA $AE,X       channel and get
D278   68          PLA             and current buffer number again
D279   20 F3 D2    JSR $D2F3       Buffer in availability map freed
D27C1  A6 82       LDX $82         Get number of current channel
D27E   B5 CD       LDA $CD,X       and corresponding buffer number
D280   C9 FF       CMP #$FF        Compare w/'buffer inactive' value
D282   F0 09       BEQ $D28D       Is the buffer used?
D284   48          PHA             YES—Save buffer number and
D285   A9 FF       LDA #$FF        buffer assignment to current
D287   95 CD       STA $CD,X       channel cleared
D289   68          PLA             Get buffer number again and
D28A   20 F3 D2    JSR $D2F3       free buffer in availability table
D28D1  60          RTS             Return from this subroutine
```

```
[CF29/CF83/D019/D20A/D21E/DC79/DD0E/F0E7]
Look for buffer
D28E   98         TYA           Get buffer number
D28F   48         PHA           and save it
D290   A0 01      LDY #$01      Look for a
D292   20 BA D2   JSR $D2BA     free buffer
D295   10 0C      BPL $D2A3     Found a buffer?
D297   88         DEY           NO-set buffer # to next buffer
D298   20 BA D2   JSR $D2BA     look for another buffer
D29B   10 06      BPL $D2A3     Found a buffer?
D29D   20 39 D3   JSR $D339     NO-get free buffer
D2A0   AA         TAX           Save buffer number
D2A1   30 13      BMI $D2B6     Has a buffer been found?
D2A3³  B5 00      LDA $00,X     YES-get last buffer jobcode
D2A5   30 FC      BMI $D2A3     Is job already running?
D2A7   A5 7F      LDA $7F       YES-get current drive number
D2A9   95 00      STA $00,X     Send return message of job loop
D2AB   9D 5B 02   STA $025B,X   and clear memory for last jobcode
D2AE   8A         TXA           Get buffer number and double  it
D2AF   0A         ASL A         (the following addresses are
D2B0   A8         TAY           passed in two-byte values)
D2B1   A9 02      LDA #$02      Buffr ptr fr start-of-data range
D2B3   99 99 00   STA $0099,Y   Set buffer pointer anew
D2B6¹  68         PLA           Re-establish buffer number and
D2B7   A8         TAY           save it
D2B8   8A         TXA           Set numer of buffers found
D2B9   60         RTS           Return from this subroutine
-----------------------------------------------------------
[D292/D298]
Look for free buffer
D2BA   A2 07      LDX #$07      Number of bits per byte -1('BPL')
D2BC¹  B9 4F 02   LDA $024F,Y   Get bit pattern of map table
D2BF   3D E9 EF   AND $EFE9,X   Get corresponding buffer bit
D2C2   F0 04      BEQ $D2C8     Is the buffer covered?
D2C4   CA         DEX           YES-set buffer countr to next bit
D2C5   10 F5      BPL $D2BC     Are all bits already tested?
D2C7   60         RTS           YES-Return from this subroutine
D2C8¹  B9 4F 02   LDA $024F,Y   Get original byte of map table &
D2CB   5D E9 EF   EOR $EFE9,X   corresponding buffer bit; set bit
D2CE   99 4F 02   STA $024F,Y   and rewrite byte
D2D1   8A         TXA           Get number of buffers found
```

```
D2D2   88        DEY          Pointer to next catalog byte
D2D3   30 03     BMI $D2D8    Both of them used?
D2D5   18        CLC          NO—calculate
D2D6   69 08     ADC #$08     new buffer number
D2D8¹  AA        TAX          Save buffer # as channel number
D2D9²  60        RTS          Return from this subroutine
--------------------------------------------------------------------
[E2BC/E2BF]
Free up all inactive buffers
D2DA   A6 82     LDX $82      Get number of current channel and
D2DC   B5 A7     LDA $A7,X    determine matching buffer
D2DE   30 09     BMI $D2E9    Is buffer occupied?
D2E0   8A        TXA          YES—get another channel number &
D2E1   18        CLC          compute for a second
D2E2   69 07     ADC #$07     buffer;
D2E4   AA        TAX          save it
D2E5   B5 A7     LDA $A7,X    Get matching buffer number
D2E7   10 F0     BPL $D2D9    Is buffer occupied?
D2E9¹  C9 FF     CMP #$FF     NO—test against'buffer free'value
D2EB   F0 EC     BEQ $D2D9    Is the buffer identified free?
D2ED   48        PHA          NO—Save buffer number
D2EE   A9 FF     LDA #$FF     Set'buffer free'value for current
D2F0   95 A7     STA $A7,X    channel
D2F2   68        PLA          call another buffer number
--------------------------------------------------------------------
[D268/D279/D28A]
Free up buffer index
D2F3   29 0F     AND #$0F     Reserve and
D2F5   A8        TAY          save
D2F6   C8        INY          buffer numbers
D2F7   A2 10     LDX #$10     Total number of buffers
D2F9¹  6E 50 02  ROR $0250    Displace buffer index
D2FC   6E 4F 02  ROR $024F    by one bit
D2FF   88        DEY          Set pointer to next buffer
D300   D0 01     BNE $D303    Any buffers left
D302   18        CLC          NO—set 'buffer free' flag
D303¹  CA        DEX          Re-establish bit index again
D304   10 F3     BPL $D2F9    Are bits back in output position?
D306   60        RTS          YES—Return from this subroutine
--------------------------------------------------------------------
```

```
[84D8/8C64/EE36]
Close channels 0-14
D307   A9 0E      LDA #$0E      Set channel number counter
D309   85 83      STA $83       to currrent secondary address
D30B1  20 27 D2   JSR $D227     and close channel
D30E   C6 83      DEC $83       Set counter to next channel #
D310   D0 F9      BNE $D30B     All channels already closed?
D312   60         RTS           YES—return from this subroutine
-------------------------------------------------------------------
[D045/EC55/EC66]
Free up all channels on current drive
D313   A9 0E      LDA #$0E      Channel number counter
D315   85 83      STA $83       Save and set channel number of
D3171  A6 83      LDX $83       current secondary address
D319   BD 2B 02   LDA $022B,X   Get corresponding status
D31C   C9 FF      CMP #$FF      & test against'channel free'value
D31E   F0 14      BEQ $D334     Is channel occupied?
D320   29 3F      AND #$3F      YES—get this channel number and
D322   85 82      STA $82       store it
D324   20 93 DF   JSR $DF93     Get buffer number and
D327   AA         TAX           save it
D328   BD 5B 02   LDA $025B,X   Get jobcode for buffer & isolate
D32B   29 01      AND #$01      the instructions from it
D32D   C5 7F      CMP $7F       Test against current drive value
D32F   D0 03      BNE $D334     Channel belong to another drive?
D331   20 27 D2   JSR $D227     NO—Free channel
D3342  C6 83      DEC $83       Counter for channel on nxt chnl
D336   10 DF      BPL $D317     All channels used?
D338   60         RTS           YES—Return from this subroutine
-------------------------------------------------------------------
[D29D]
Get a free buffer
D339   A5 6F      LDA $6F       Get channel number
D33B   48         PHA           and save it
D33C   A0 00      LDY #$00      Set chnl # cntr to start value
D33E1  B6 FA      LDX $FA,Y     Get number of channel
D340   B5 A7      LDA $A7,X     Get number of buffer assigned
D342   10 04      BPL $D348     Is buffer being used?
D344   C9 FF      CMP #$FF      NO—test against'buffer free'value
D346   D0 16      BNE $D35E     Is buffer free?
D3482  8A         TXA           YES—Get another channel number
D349   18         CLC           and convert for access to
D34A   69 07      ADC #$07      a second buffer;
D34C   AA         TAX           save it
D34D   B5 A7      LDA $A7,X     Get corresponding buffer
D34F   10 04      BPL $D355     Is buffer occupied?
```

```
D351   C9 FF       CMP #$FF         NO—test against val:'buffer free'
D353   D0 09       BNE $D35E        Is buffer free?
D355²  C8          INY              YES—choose next channel
D356   C0 05       CPY #$05         Compare with max. # of channels
D358   90 E4       BCC $D33E        Are all channels worked with?
D35A   A2 FF       LDX #$FF         Error flag value
D35C   D0 1C       BNE $D37A        Jump to $D37A
D35E²  86 6F       STX $6F          Set channel number;
D360   29 3F       AND #$3F         use to determine buffer number
D362   AA          TAX              and save it
D363¹  B5 00       LDA $00,X        Get jobcode of buffer
D365   30 FC       BMI $D363        Is job still in process?
D367   C9 02       CMP #$02         NO—test return msg against 'OK'
D369   90 08       BCC $D373        Job run error-free?
D36B   A6 6F       LDX $6F          NO—get channel number and test
D36D   E0 07       CPX #$07         for maximum number
D36F   90 D7       BCC $D348        Channel number in allowed range?
D371   B0 E2       BCS $D355        NO—Jump to $D355
D373¹  A4 6F       LDY $6F          Get channel number and label
D375   A9 FF       LDA #$FF         buffer in buffer assignment table
D377   99 A7 00    STA $00A7,Y      as free
D37A¹  68          PLA              Get originl channel number again
D37B   85 6F       STA $6F          and reset it
D37D   8A          TXA              Give buffer number
D37E   60          RTS              Return from this subroutine
-----------------------------------------------------------------------
[D1E9]
Seek and lay out free channel
D37F   A0 00       LDY #$00         Initialize pointers
D381   A9 01       LDA #$01         Bit of channel to be tested
D383¹  2C 56 02    BIT $0256        Test bit in channel catalog
D386   D0 09       BNE $D391        Is channel free?
D388   C8          INY              NO—pick next channel
D389   0A          ASL A            Bit positioned for next channel
D38A   D0 F7       BNE $D383        Have all channels been checked?
D38C   A9 70       LDA #$70         YES—error message
D38E   4C C8 C1    JMP $C1C8        '70 No Channel' displayed
D391¹  49 FF       EOR #$FF         Invert 'channel free' bitflag and
D393   2D 56 02    AND $0256        focus down into flag byte
D396   8D 56 02    STA $0256        Lay out channel
D399   98          TYA              Get channel number
D39A   60          RTS              Return from this subroutine
-----------------------------------------------------------------------
```

```
[CA39]
Get byte from channel
D39B  20 EB D0   JSR $D0EB      Open read channel
D39E  20 00 C1   JSR $C100      Switch on LED of current drive
D3A1  20 AA D3   JSR $D3AA      Read out byte over channel
D3A4  A6 82      LDX $82        Determine channel number
D3A6  BD 3E 02   LDA $023E,X    and get corresponding data byte
D3A9  60         RTS            Return from this subroutine
--------------------------------------------------------------
[82BD/D3A1/E992]
Read byte from file
D3AA  A6 82      LDX $82        Get channel number
D3AC  20 25 D1   JSR $D125      Determine filetype
D3AF  D0 03      BNE $D3B4      Is it a relative file?
D3B1  4C 20 E1   JMP $E120      YES—REL file routine
D3B4¹ A5 83      LDA $83        Get secondary address and
D3B6  C9 0F      CMP #$0F       compare with command channel (15)
D3B8  F0 5A      BEQ $D414      Command channel produced?
D3BA  B5 F2      LDA $F2,X      NO—Get  channel status and
D3BC  29 08      AND #$08       test for EOI flag
D3BE  D0 13      BNE $D3D3      Was last byte transferred?
D3C0  20 25 D1   JSR $D125      YES—Determine filetype & compare
D3C3  C9 07      CMP #$07       with value for direct access
D3C5  D0 07      BNE $D3CE      Direct access chanel been opened?
D3C7  A9 89      LDA #$89       YES—Send direct access flag value
D3C9  95 F2      STA $F2,X      as channel status
D3CB  4C DE D3   JMP $D3DE      Get byte from buffer
D3CE¹ A9 00      LDA #$00       Flag value for EOI encountered;
D3D0  95 F2      STA $F2,X      close channel and clear map
D3D2  60         RTS            Return from this subroutine
D3D3¹ A5 83      LDA $83        Get current secondary address
D3D5  F0 32      BEQ $D409      Should it be loaded as a program?
D3D7  20 25 D1   JSR $D125      NO—Determine filetype and
D3DA  C9 04      CMP #$04       compare w/value for relative file
D3DC  90 22      BCC $D400      Identical?
--------------------------------------------------------------
[D3CB/FFB0]
Get byte from relative file
D3DE  20 2F D1   JSR $D12F      YES—Set buffer & channel numbers
D3E1  B5 99      LDA $99,X      Get current buffr pointer,compare
D3E3  D9 44 02   CMP $0244,Y    with the end-of-buffer
D3E6  D0 04      BNE $D3EC      End of effective range reached
D3E8  A9 00      LDA #$00       Buffer pointer (low-byte)
D3EA  95 99      STA $99,X      reset
--------------------------------------------------------------
```

```
[CD59/D3E6]
Get next byte from file
D3EC   F6 99       INC $99,X        Set buffer pointer to next byte
-------------------------------------------------------------------
[CDCA/CFD5]
Get currrent byte from file
D3EE   A1 99       LDA ($99,X)      Read byte from buffer and save
D3F0   99 3E 02    STA $023E,Y      as byte to be given
D3F3   B5 99       LDA $99,X        Get buffer pointer and
D3F5   D9 44 02    CMP $0244,Y      test against end value
D3F8   D0 05       BNE $D3FF        Reached end of the file range?
D3FA   A9 81       LDA #$81         YES—Take flag value for 'last
D3FC   99 F2 00    STA $00F2,Y      char into channel status table
D3FF1  60          RTS              Return from this subroutine
D4002  20 56 D1    JSR $D156        Get character from buffer
D4031  A6 82       LDX $82          Get number of current channel and
D405   9D 3E 02    STA $023E,X      allocate databyte for output
D408   60          RTS              Return from this subroutine
D4091  AD 54 02    LDA $0254        Get flag for directory
D40C   F0 F2       BEQ $D400        Is directory in buffer?
D40E   20 67 ED    JSR $ED67        YES—Get byte from directory
D411   4C 03 D4    JMP $D403        and take it over
-------------------------------------------------------------------
[D3B8]
Read error channel
D414   20 E8 D4    JSR $D4E8        Get current buffer pointer
D417   C9 D4       CMP #$D4         Compare w/ error buffer value
D419   D0 18       BNE $D433        Is  the pointer properly set?
D41B   A5 95       LDA $95          YES—Get pointer hi-byte and test
D41D   C9 02       CMP #$02         against correct value
D41F   D0 12       BNE $D433        Is pointr directd at error buffr?
D421   A9 0D       LDA #$0D         YES—'Return'
D423   85 85       STA $85          Output to next byte
D425   20 23 C1    JSR $C123        Reset error flag
D428   A9 00       LDA #$00         Number of 'OK' message
D42A   20 C1 E6    JSR $E6C1        Write message to error buffer
D42D   C6 A5       DEC $A5          Pointer to errormessge buffr (lo)
D42F   A9 80       LDA #$80         'Read' flag
D431   D0 12       BNE $D445        Jump to $D445
D4332  20 37 D1    JSR $D137        Get byte from error buffer and
D436   85 85       STA $85          take as byte to be output
D438   D0 09       BNE $D443        Reached the end?
-------------------------------------------------------------------
```

```
[CB48]
Set pointer for error message pointer($02D4)
D43A  A9 D4      LDA #$D4         YES—Set buffer pointer for
D43C  20 C8 D4   JSR $D4C8        error buffer
D43F  A9 02      LDA #$02         Set pointer
D441  95 9A      STA $9A,X        high-byte
----------------------------------------------------------------
[CB42/D438]
Initialize error message channel
D443  A9 88      LDA #$88         Set 'read' and 'EOI'
D445¹ 85 F7      STA $F7          flags for channel 5
D447  A5 85      LDA $85          Get byte and take up
D449  8D 43 02   STA $0243        output
D44C  60         RTS              Return from this subroutine
----------------------------------------------------------------
[C629/C8B0/EE07]
Read next sector of a file
D44D  20 93 DF   JSR $DF93        Determine buffer number
D450  0A         ASL A            and double it
D451  AA         TAX              (pointr table works w/2-byte #'s)
D452  A9 00      LDA #$00         Starting position in buffer
D454  95 99      STA $99,X        taken up in buffr pointr (lobyte)
D456  A1 99      LDA ($99,X)      Get track number of next sector
D458  F0 05      BEQ $D45F        No more sectors on hand
D45A  D6 99      DEC $99,X        Set buffer pointer to end
D45C  4C 56 D1   JMP $D156        Read next sector
D45F¹ 60         RTS              Return from this subroutine
----------------------------------------------------------------
[CD39/D720/DBC9]
Take jobcode for 'read sector'
D460  A9 80      LDA #$80         Set jobcode for 'read sector'
D462  D0 02      BNE $D466        Jump to $D466
----------------------------------------------------------------
[CD8C/CD9D/D790/D93A/D98A/EEAC]
Take up jobcode for 'write sector'
D464  A9 90      LDA #$90         Set jobcode  for 'write sector'
----------------------------------------------------------------
[D462]
Execute jobcode (in A)
D466  05 7F      ORA $7F          Set current drive # in jobcode
D468  8D 4D 02   STA $024D        and save jobcode
D46B  A5 F9      LDA $F9          Get number of current buffer
D46D  20 D3 D6   JSR $D6D3        Take track and sector numbers
D470  A6 F9      LDX $F9          Get number of current buffer
D472  4C 93 D5   JMP $D593        Set jobcode and execute job
----------------------------------------------------------------
```

```
[C5C1/C60E/C880/CA0C/EDEB]
Open sequential file for reading
D475  A9 01      LDA #$01        Filetype identifier for SEQ file
-------------------------------------------------------------------
[E7D5]
Open file for reading
D477  8D 4A 02   STA $024A       determined
D47A  A9 11      LDA #$11        # of internal read channels (17)
D47C  85 83      STA $83         taken as current 2ndary address
D47E  20 46 DC   JSR $DC46       Buffer laid out & sector read in
D481  A9 02      LDA #$02        Buffer pointer set to start of
D483  4C C8 D4   JMP $D4C8       file range
-------------------------------------------------------------------
[C9B0]
Open file for  writing
D486  A9 12      LDA #$12        Set # of internal write channel
D488  85 83      STA $83         (18) set as secondary address
D48A  4C DA DC   JMP $DCDA       Open channel & laydown new sector
-------------------------------------------------------------------
[D730]
Write next directory sector
D48D  20 3B DE   JSR $DE3B       Get current track/sector numbers
D490  A9 01      LDA #$01        Number of sectors to
D492  85 6F      STA $6F         be laid down
D494  A5 69      LDA $69         Get normal sector set and
D496  48         PHA             retain
D497  A9 03      LDA #$03        Declare sector set for directory
D499  85 69      STA $69         at 3
D49B  20 2D F1   JSR $F12D       Transmit next free sector
D49E  68         PLA             Re-direct normal
D49F  85 69      STA $69         sector set
D4A1  A9 00      LDA #$00        Set buffer pointer to
D4A3  20 C8 D4   JSR $D4C8       start-of-buffer
D4A6  A5 80      LDA $80         Write track # of new sector in
D4A8  20 F1 CF   JSR $CFF1       current directory sector
D4AB  A5 81      LDA $81         Take # of next sector in current
D4AD  20 F1 CF   JSR $CFF1       sector as string
D4B0  20 C7 D0   JSR $D0C7       Write current sector to diskette
D4B3  20 99 D5   JSR $D599       Wait until job loop is ready
D4B6  A9 00      LDA #$00        Reset buffer pointer
D4B8  20 C8 D4   JSR $D4C8       to beginning
D4BB[1] 20 F1 CF JSR $CFF1       Write fillbytes into buffer
D4BE  D0 FB      BNE $D4BB       Entire buffer cleared?
D4C0  20 F1 CF   JSR $CFF1       YES—Identifier for last sector
D4C3  A9 FF      LDA #$FF        Write number of good sector bytes
D4C5  4C F1 CF   JMP $CFF1       in sector
-------------------------------------------------------------------
```

```
[C614/C896/CA14/CA32/CAB4/CD47/CD83/D16F/D1AB/D1C3/D43C/D483/D4A3/D4B8]
[D740/D914/DA42/DB92/DCA0/DD6F/DE97/DFFA/E04F/E27A/E476/E4A3/E4C0/E4DB]
[ECA9/EDF0]
Set buffer pointer to given position
D4C8  85 6F     STA $6F        Save new position
D4CA  20 93 DF  JSR $DF93      Get current buffer number and
D4CD  0A        ASL A          double it (pointer table takes
D4CE  AA        TAX            2-byte pointers)
D4CF  B5 9A     LDA $9A,X      Get and set
D4D1  85 95     STA $95        buffer pointer (high-byte)
D4D3  A5 6F     LDA $6F        Get low-byte of buffer pointer
D4D5  95 99     STA $99,X      Save and set as current
D4D7  85 94     STA $94        buffer pointer
D4D9  60        RTS            Return from this subroutine
----------------------------------------------------------------
[C1BA/DAD1/E653]
Close internal channels
D4DA  A9 11     LDA #$11       Set # of internal read channel
D4DC  85 83     STA $83        (17) as current secondary address
D4DE  20 27 D2  JSR $D227      Close channel
D4E1  A9 12     LDA #$12       Store number of internal write
D4E3  85 83     STA $83        channel(18)as current 2ndry adrs;
D4E5  4C 27 D2  JMP $D227      Close channel
----------------------------------------------------------------
[C5D7/C6E5/CD76/CFDC/D1C8/D414/DB6A/DB76/DFEA/E182/E1A9]
Determine current buffer pointer
D4E8  20 93 DF  JSR $DF93      Get number of current buffer
----------------------------------------------------------------
[DF49]
Set buffer pointer (buffer number in A)
D4EB  0A        ASL A          Double it (pointer table deals
D4EC  AA        TAX            with 2-byte  numbers)
D4ED  B5 9A     LDA $9A,X      Get pointer at position in buffer
D4EF  85 95     STA $95        and take on as
D4F1  B5 99     LDA $99,X      current
D4F3  85 94     STA $94        buffer pointer
D4F5  60        RTS            Return from this subroutine
----------------------------------------------------------------
[C5D1/CF39/CF40/E00E/E39F]
Read any byte from buffer
(A must contain position of the character)
D4F6  85 71     STA $71        Save buffer position
D4F8  20 93 DF  JSR $DF93      Determine current buffer number
D4FB  AA        TAX            and save it
D4FC  BD E0 FE  LDA $FEE0,X    Get hi-byte of appropriate buffer
D4FF  85 72     STA $72        address and set it
D501  A0 00     LDY #$00       Initialize buffer pointer and
```

```
D503   B1 71       LDA ($71),Y      get byte from buffer position
D505   60          RTS              Return from this subroutine
-------------------------------------------------------------
[D0D0/DE32]
Test track and sector numbers for validity, then set jobcode
D506   BD 5B 02    LDA $025B,X      Get jobcode declared by buffr and
D509   29 01       AND #$01         determine drive number from it
D50B   0D 4D 02    ORA $024D        Concentrate on markiing
D50E1  48          PHA              current jobcode
D50F   86 F9       STX $F9          Hold on to buffer number
D511   8A          TXA              and double
D512   0A          ASL A            the number (the next table uses
D513   AA          TAX              2-byte values)
D514   B5 07       LDA $07,X        Get sector number of this job
D516   8D 4D 02    STA $024D        and save it
D519   B5 06       LDA $06,X        determine track # of this job
D51B   F0 2D       BEQ $D54A        No track chosen (0)?
D51D   CD AC 02    CMP $02AC        NO—and test for largest track +1
D520   B0 28       BCS $D54A        Is this track # in allowed range?
D522   AA          TAX              YES—Save track number
D523   68          PLA              and call jobcode back;
D524   48          PHA              save it again
D525   29 F0       AND #$F0         Isolate jobcode
D527   C9 90       CMP #$90         and compare with code for 'write'
D529   D0 4F       BNE $D57A        Identical?
D52B   68          PLA              YES—get entire jobcode again and
D52C   48          PHA              save it immediately
D52D   4A          LSR A            Bitflag for drive number in Carry
D52E   B0 05       BCS $D535        Drive 1 chosen?
D530   AD 01 01    LDA $0101        NO—Format identifier for drive 0
D533   90 03       BCC $D538        Jump to $D538
D5351  AD 02 01    LDA $0102        Get format identifier for drive 1
D5381  F0 05       BEQ $D53F        Jump to $D53F
D53A   CD D5 FE    CMP $FED5        Compare with identifier 'A'
D53D   D0 33       BNE $D572        Right format?
D53F1  8A          TXA              YES—get track number again
D540   20 4B F2    JSR $F24B        Get largest appropriate sector #
D543   CD 4D 02    CMP $024D        Compare with sector number of job
D546   F0 02       BEQ $D54A        Reached the maximum number?
D548   B0 30       BCS $D57A        NO—is sector number legal?
D54A3  20 52 D5    JSR $D552        NO—get track and sector of job
D54D4  A9 66       LDA #$66         again, and display error message
D54F   4C 45 E6    JMP $E645        '66 Illegal Track or Sector'
-------------------------------------------------------------
```

```
[D54A/D572]
Get track/sector of current job from job memory
D552  A5 F9     LDA $F9       Get # of current job (buffer)
D554  0A        ASL A         and double
D555  AA        TAX           (table works w/ 2-byte values)
D556  B5 06     LDA $06,X     Get job track # from table and
D558  85 80     STA $80       save as current track
D55A  B5 07     LDA $07,X     Get job sector number & store as
D55C  85 81     STA $81       current sector number
D55E  60        RTS           Return from this subroutine
-----------------------------------------------------------------
[CE08/EDE5]
Check current track/sector for allowable range
D55F  A5 80     LDA $80       Get current track number
D561  F0 EA     BEQ $D54D     No track set?
D563  CD AC 02  CMP $02AC     NO—Test for max. allowable tracks
D566  B0 E5     BCS $D54D     Allowable track number (< max.)?
D568  20 4B F2  JSR $F24B     YES—Get # of sectors in track, &
D56B  C5 81     CMP $81       compare with current sector #
D56D  F0 DE     BEQ $D54D     Is the sector number 1 too high?
D56F  90 DC     BCC $D54D     NO—Is the number still larger?
D571  60        RTS           NO—Return from this subroutine
-----------------------------------------------------------------
[D53D/EE53]
Display error message for false format
D572  20 52 D5  JSR $D552     Get track/sector of job and
D575  A9 73     LDA #$73      display error message --
D577  4C 45 E6  JMP $E645     '73 CBM DOS V3.0 1571'
-----------------------------------------------------------------
[D529/D548]
Send job for current buffer to job loop
(NB:Routine cannot jump with 'JSR', since the stack must contain
the jobcode and not the jump address)
D57A  A6 F9     LDX $F9       Get the # of the current buffer
D57C  68        PLA           get jobcode to be set and
D57D  8D 4D 02  STA $024D     store as current jobcode
D580  95 00     STA $00,X     Give to job loop
D582  9D 5B 02  STA $025B,X   Assign to current buffer
D585  60        RTS           Return from this subroutine
-----------------------------------------------------------------
[A5D1/A66E/A693/A6BA]
Send jobcode for read to job loop and wait until execution
D586  A9 80     LDA #$80      Jobcode for 'read sector'
D588  D0 02     BNE $D58C     Jump to $D58C
-----------------------------------------------------------------
```

```
[A594/A5A4/A5C5]
Send jobcode for write to job loop, and wait until execution
D58A  A9 90      LDA #$90        Jobcode for 'write sector'
------------------------------------------------------------------
[A6E5/A70E/D588]
Execute job for current drive (jobcode in A)
D58C  05 7F      ORA $7F         Take  current drive in jobcode
D58E  A6 F9      LDX $F9         Get number of proper buffers
------------------------------------------------------------------
[DC3D]
Execute  jobcode (jobcode in A, buffer number in X)
D590  8D 4D 02   STA $024D       and save current jobcode
------------------------------------------------------------------
[D472/DF42]  Execute job
D593  AD 4D 02   LDA $024D       Get jobcode; test track/sector
D596  20 0E D5   JSR $D50E       parameters; and wait in job loop,
------------------------------------------------------------------
[869A/C8BE/CAAC/CAC6/D0A1/D4B3/DB9F/DC95/DD6A/DD84/DDF9/E068/E430/E4A9]
[E4F0/CF73/E05A]
Wait until job is executed and error message  is prepared
D599  20 A6 D5   JSR $D5A6       Control job run
D59C  B0 FB      BCS $D599       Is job finished yet?
D59E  48         PHA             YES—Save return message of job
D59F  A9 00      LDA #$00        Clear 'Error from job'
D5A1  8D 98 02   STA $0298       flag and
D5A4  68         PLA             get return message again
D5A5  60         RTS             Return from this subroutine
------------------------------------------------------------------
[D599]
Supervise current job run
D5A6  B5 00      LDA $00,X       Get jobcode from job  memory
D5A8  30 1A      BMI $D5C4       Is job still in process?
D5AA  C9 02      CMP #$02        NO—Test for 'OK' message
D5AC  90 14      BCC $D5C2       Job properly run?
D5AE  C9 08      CMP #$08        NO—Compare w/ 'Write Protect On'
D5B0  F0 08      BEQ $D5BA       Is  write-protect notch covered?
D5B2  C9 0B      CMP #$0B        NO—Compare w/ 'Disk ID Mismatch'
D5B4  F0 04      BEQ $D5BA       Find a false ID?
D5B6  C9 0F      CMP #$0F        NO—Compare w/ 'Drive Not Ready'
D5B8  D0 0C      BNE $D5C6       Unformatted diskette in drive?
D5BA2 2C 98 02   BIT $0298       YES—Test error flag
D5BD  30 03      BMI $D5C2       Has an error been displayed?
D5BF  4C 3F D6   JMP $D63F       NO—Display error message
D5C22 18         CLC             Set flag for 'Job finished'
D5C3  60         RTS             Return from this subroutine
D5C41 38         SEC             Set flg f/'Job not finished yet'
D5C5  60         RTS             Return from this subroutine
```

```
[D5B8/D644:A6CE]
Set head to next track after a read error; search some more
D5C6  98        TYA          Reserve Y-Register
D5C7  48        PHA          (routine will change it)
D5C8  A5 7F     LDA $7F      Get current drive  number and
D5CA  48        PHA          save it
D5CB  BD 5B 02  LDA $025B,X  Get buffer-declared jobcode and
D5CE  29 01     AND #$01     determine drive used
D5D0  85 7F     STA $7F      Store # of current drive and get
D5D2  A8        TAY          bitmask stated by drive,
D5D3  B9 CA FE  LDA $FECA,Y  to switch drive LED on
D5D6  8D 6D 02  STA $026D    Save LED-blink mask
D5D9  20 A6 D6  JSR $D6A6    ($6A) Execute read-search
D5DC  C9 02     CMP #$02     Compare return message w/ 'OK'
D5DE  B0 03     BCS $D5E3    Last job run without errors?
D5E0  4C 6D D6  JMP $D66D    YES—End of routine
D5E3¹ BD 5B 02  LDA $025B,X  Get current jobcode
D5E6  29 F0     AND #$F0     Isolate and save
D5E8  48        PHA          command bits
D5E9  C9 90     CMP #$90     Compare with value for 'write'
D5EB  D0 07     BNE $D5F4    Has a sector been written?
D5ED  A5 7F     LDA $7F      YES—Get drive number and set
D5EF  09 B8     ORA #$B8     jobcode for 'look for sector'
D5F1  9D 5B 02  STA $025B,X  Assign jobcode to current buffer
D5F4¹ 24 6A     BIT $6A      Flg fr'don't look for next track'
D5F6  70 39     BVS $D631    Is flag set?
D5F8  A9 00     LDA #$00     NO—Initialize pointers:
D5FA  8D 99 02  STA $0299    Position pointer to next track
D5FD  8D 9A 02  STA $029A    Pointer to searchphase—next track
D600¹ AC 99 02  LDY $0299    Determine positioning phase
D603  AD 9A 02  LDA $029A    Get currnt cntrl byt f/head move-
D606  38        SEC          ment and sent value for return to
D607  F9 DB FE  SBC $FEDB,Y  outside position, then
D60A  8D 9A 02  STA $029A    positioning next to the track
D60D  B9 DB FE  LDA $FEDB,Y  Get cntrl byt for 1/2step to next
D610  20 A1 FF  JSR $FFA1    track; execute head movement
D613  EE 99 02  INC $0299    Set counter to next control byte
D616  20 A6 D6  JSR $D6A6    ($6A) Execute read search
D619  C9 02     CMP #$02     Test retrn messge aganst'OK'value
D61B  90 08     BCC $D625    Any errors?
D61D  AC 99 02  LDY $0299    Get counter for positioning phase
D620  B9 DB FE  LDA $FEDB,Y  Get next positioning command
D623  D0 DB     BNE $D600    End of search string?
D625¹ AD 9A 02  LDA $029A    YES—Get cntrl value for return to
D628  20 A6 FF  JSR $FFA6    track & look for a reading again
D62B  B5 00     LDA $00,X    Get return value of job  loop and
D62D  C9 02     CMP #$02     compare with 'Ok'
```

```
D62F   90 2B     BCC $D65C      Read-search go well?
D631¹  24 6A     BIT $6A        NO—Check flag:'head at track 0'
D633   10 0F     BPL $D644      Re-adjust head (Bump) ?
D635¹  68        PLA            NO—Get command code,test against
D636   C9 90     CMP #$90       'write sector' job
D638   D0 05     BNE $D63F      Identical?
D63A   05 7F     ORA $7F        YES—Set drive #, assign current
D63C   9D 5B 02  STA $025B,X    buffer a jobcode
D63F³  B5 00     LDA $00,X      Get return message of job
D641   20 0A E6  JSR $E60A      and prep error message
D644²  68        PLA            Get proper command code
D645   2C 98 02  BIT $0298      Test error flag
D648   30 23     BMI $D66D      Found an error already?
D64A   48        PHA            NO—Save jobcode and
D64B   A9 C0     LDA #$C0       set jobcode for 'head re-
D64D   05 7F     ORA $7F        adjusted' (Bump) for current
D64F   95 00     STA $00,X      drive
D651   20 B6 9F  JSR $9FB6      Start job loop and execute job
D654   EA        NOP            [via modification of 1541 ROM]
D655   20 A6 D6  JSR $D6A6      Job executed ($6A) times
D658   C9 02     CMP #$02       Compare return message w/ 'OK'
D65A   B0 D9     BCS $D635      Was this last run correctly?
D65C¹  68        PLA            YES—Get jobcode again and compare
D65D   C9 90     CMP #$90       with value for 'write'
D65F   D0 0C     BNE $D66D      Should sector have been written?
D661   05 7F     ORA $7F        YES—Set drive number and assign
D663   9D 5B 02  STA $025B,X    jobcode to current buffer
D666   20 A6 D6  JSR $D6A6      ($6A) times—look for sector write
D669   C9 02     CMP #$02       Compare return mess. w/ 'OK'
D66B   B0 D2     BCS $D63F      Successful write?
D66D³  68        PLA            YES—Prep current drive number
D66E   85 7F     STA $7F        again
D670   68        PLA            Reset
D671   A8        TAY            Y-register
D672   B5 00     LDA $00,X      Get return message for job loop
D674   18        CLC            Set flag for 'Job finished'
D675   60        RTS            Return from this subroutine
---------------------------------------------------------------------
[FF99/FF9C]
Accumulator instructs head to move in half-steps
(Bit7 =1 step in; Bit7 =0 step out)
D676   C9 00     CMP #$00       Test contents of accumulator
D678   F0 18     BEQ $D692      Step value given?
D67A   30 0C     BMI $D688      YES—Should head move out?
D67C¹  A0 01     LDY #$01       Value for half-step in
D67E   20 93 D6  JSR $D693      Reset head
D681   38        SEC            and decrement counter
```

```
D682   E9 01       SBC #$01        for number of half-steps
D684   D0 F6       BNE $D67C       All steps taken?
D686   F0 0A       BEQ $D692       YES—Jump to $D692
D688²  A0 FF       LDY #$FF        Value for half-step out
D68A   20 93 D6    JSR $D693       Set head again and
D68D   18          CLC             increment counter for
D68E   69 01       ADC #$01        number of half-steps
D690   D0 F6       BNE $D688       All steps covered?
D692²  60          RTS             YES—Return from this subroutine
-------------------------------------------------------------------
[D67E/D68A]
Head movement values given to job loop
D693   48          PHA             Reserve accumulator
D694   98          TYA             Get value for head positioning
D695   A4 7F       LDY $7F         Get current drive number and
D697   99 FE 02    STA $02FE,Y     send control byte to job loop
D69A¹  D9 FE 02    CMP $02FE,Y     Get value again
D69D   F0 FB       BEQ $D69A       Was value taken and head set?
D69F   A9 00       LDA #$00        YES—clear
D6A1   99 FE 02    STA $02FE,Y     job register
D6A4   68          PLA             Re-establish accumulator
D6A5   60          RTS             Return from this subroutine
-------------------------------------------------------------------
0[D5D9/D616/D655/D666]
Jobcode executes until successful, or when counter in $6A=0
D6A6   A5 6A       LDA $6A         Get search number and limit
D6A8   29 3F       AND #$3F        to a range of 0 to 63
D6AA   A8          TAY             Set counter
D6AB¹  AD 6D 02    LDA $026D       Switch on LED mask
D6AE   4D 00 1C    EOR $1C00       LED bit in drive control register
D6B1   8D 00 1C    STA $1C00       switches (LED flickers)
D6B4   BD 5B 02    LDA $025B,X     Get jobcode of current buffer and
D6B7   95 00       STA $00,X       send to job loop
D6B9   20 B6 9E    JSR $9EB6       Start job loop and execute job
D6BC   EA          NOP             [1541 ROM modification]
D6BD   C9 02       CMP #$02        Compare return message w/ 'OK'
D6BF   90 03       BCC $D6C4       Is job completed?
D6C1   88          DEY             NO—decrement trial number
D6C2   D0 E7       BNE $D6AB       Any more trials to be done?
D6C4¹  48          PHA             NO—Save job number
D6C5   AD 6D 02    LDA $026D       Get 'LED on' mask and
D6C8   0D 00 1C    ORA $1C00       concentrate remaining bits of
D6CB   8D 00 1C    STA $1C00       contrl registers; set registers
D6CE   68          PLA             Get return message from job loop
D6CF   60          RTS             Return from this subroutine
-------------------------------------------------------------------
```

```
[D09B/D0BA/D186/DCE2/DE7F/E3B9/E3CB]
Send current track & sector number to job loop
D6D0  20 93 DF   JSR $DF93       Get current buffer number
---------------------------------------------------------------
[A414/C8D7/D03A/D46D/DC8F/DD2E/DF3D]
Send track & sector to job loop (buffer in A)
D6D3  0A         ASL A           and double (job table works
D6D4  A8         TAY             with 2-byte values)
D6D5  A5 80      LDA $80         Send number of current track
D6D7  99 06 00   STA $0006,Y     to job loop
D6DA  A5 81      LDA $81         Store current sector number
D6DC  99 07 00   STA $0007,Y     for job loop
D6DF  A5 7F      LDA $7F         Get current drive number
D6E1  0A         ASL A           double and
D6E2  AA         TAX             save
D6E3  60         RTS             Return from this subroutine
---------------------------------------------------------------
[C9B3/D9EC] Close file entry in directory
D6E4  A5 83      LDA $83         Get and retain current secondary
D6E6  48         PHA             address
D6E7  A5 82      LDA $82         Get and retain current channel
D6E9  48         PHA             number
D6EA  A5 81      LDA $81         Get and retain current sector
D6EC  48         PHA             number
D6ED  A5 80      LDA $80         Get and retain current track
D6EF  48         PHA             number
D6F0  A9 11      LDA #$11        Set # of internal read channel
D6F2  85 83      STA $83         (17) as current channel number
D6F4  20 3B DE   JSR $DE3B       Determine track & sector number
D6F7  AD 4A 02   LDA $024A       Get and hold on to
D6FA  48         PHA             current filetype
D6FB  A5 E2      LDA $E2         Produce drive number of new file
D6FD  29 01      AND #$01        and establish
D6FF  85 7F      STA $7F         as current drive number
D701  A6 F9      LDX $F9         Current buffer number
D703  5D 5B 02   EOR $025B,X     Get drive # belonging to jobcode
D706  4A         LSR A           and compare with actual drive no.
D707  90 0C      BCC $D715       Are the two drives identical?
D709  A2 01      LDX #$01        Set pointer to
D70B  8E 92 02   STX $0292       applicable entry
D70E  20 AC C5   JSR $C5AC       Look for next free entry
D711  F0 1D      BEQ $D730       All of them covered?
D713  D0 28      BNE $D73D       NO—Write entry to directory
D715[1] AD 91 02 LDA $0291       Get number of directory sector
D718  F0 0C      BEQ $D726       Sector number set?
D71A  C5 81      CMP $81         YES—compare with current sector #
D71C  F0 1F      BEQ $D73D       Identical?
```

```
D71E   85 81      STA $81       NO—Get number of current sector
D720   20 60 D4   JSR $D460     Read sector into buffer
D723   4C 3D D7   JMP $D73D     Put out new entry
D726¹  A9 01      LDA #$01      Set pointer to appropriate
D728   8D 92 02   STA $0292     file entry
D72B   20 17 C6   JSR $C617     Get last sector of directory
D72E   D0 0D      BNE $D73D     Any entries still free?
D730¹  20 8D D4   JSR $D48D     NO—Lay out new directory sector
D733   A5 81      LDA $81       Get sector number and set
D735   8D 91 02   STA $0291     in pointer for directory sectors
D738   A9 02      LDA #$02      Initialize buffer pointer to
D73A   8D 92 02   STA $0292     start of file range
D73D⁴  AD 92 02   LDA $0292     Current pointer position
D740   20 C8 D4   JSR $D4C8     Set buffer pointer
D743   68         PLA           Get back current filetype
D744   8D 4A 02   STA $024A     and set again
D747   C9 04      CMP #$04      Compare w/ relative file value
D749   D0 02      BNE $D74D     Is it a relative file?
D74B   09 80      ORA #$80      YES—File recognized as closed
D74D¹  20 F1 CF   JSR $CFF1     Enter filetype into directory
D750   68         PLA           Get track # of the first file
D751   8D 80 02   STA $0280     sector again; store it
D754   20 F1 CF   JSR $CFF1     Write track number into directory
D757   68         PLA           Get # of first sector of the file
D758   8D 85 02   STA $0285     and save it
D75B   20 F1 CF   JSR $CFF1     Write sector number to directory
D75E   20 93 DF   JSR $DF93     Get # of directory buffer and
D761   A8         TAY           note it
D762   AD 7A 02   LDA $027A     Fetch and save filename position
D765   AA         TAX           in input buffer
D766   A9 10      LDA #$10      Length of filename
D768   20 6E C6   JSR $C66E     Write filename to directory
D76B   A0 10      LDY #$10      Buffer pointer to end-of-filename
D76D   A9 00      LDA #$00      Write empty bytes to buffer--fill
D76F¹  91 94      STA ($94),Y   out filename
D771   C8         INY           Buffer pointer to next byte
D772   C0 1B      CPY #$1B      Compare pointer with end value
D774   90 F9      BCC $D76F     Entire buffer filled?
D776   AD 4A 02   LDA $024A     YES—Get current filetype
D779   C9 04      CMP #$04      Compare w/value for relative file
D77B   D0 13      BNE $D790     Is a relative file being opened?
D77D   A0 10      LDY #$10      YES—Buffer pointer to end of name
D77F   AD 59 02   LDA $0259     Get track # of first side-sector
D782   91 94      STA ($94),Y   and write into entry
D784   C8         INY           Buffer pointer to next position
D785   AD 5A 02   LDA $025A     Get sector #, write in directory
D788   91 94      STA ($94),Y   buffer
```

```
D78A    C8          INY              Buffer pointer to next byte
D78B    AD 58 02    LDA $0258        Get record lengtth and write
D78E    91 94       STA ($94),Y      in directory
D790[1] 20 64 D4    JSR $D464        Write directory sector to disk
D793    68          PLA              Get current channel # and set
D794    85 82       STA $82          again
D796    AA          TAX              Save channel number
D797    68          PLA              Get current 2ndary address back
D798    85 83       STA $83          and set it
D79A    AD 91 02    LDA $0291        Get track # of file entry and
D79D    85 D8       STA $D8          save; put
D79F    9D 60 02    STA $0260,X      in buffer
D7A2    AD 92 02    LDA $0292        Get sector number of file entry
D7A5    85 DD       STA $DD          and save it
D7A7    9D 66 02    STA $0266,X      Put number into directory buffer
D7AA    AD 4A 02    LDA $024A        Get filetype and
D7AD    85 E7       STA $E7          save it
D7AF    A5 7F       LDA $7F          Get current drive number and
D7B1    85 E2       STA $E2          include in file entry
D7B3    60          RTS              Return from this subroutine
----------------------------------------------------------------
[BF5D/C15D]
Take on OPEN command with secondary addresses 0-14
D7B4    A5 83       LDA $83          Get current secondary address
D7B6    8D 4C 02    STA $024C        and save it
D7B9    20 B3 C2    JSR $C2B3        Set pointer for command string
D7BC    8E 2A 02    STX $022A        Clear command channel number (0)
D7BF    AE 00 02    LDX $0200        Get first char in input buffer
D7C2    AD 4C 02    LDA $024C        Get secondary address
D7C5    D0 2C       BNE $D7F3        Is there a LOAD command?
D7C7    E0 2A       CPX #$2A         YES—Check for '*' as 1st char
D7C9    D0 28       BNE $D7F3        First file entry loaded?
D7CB    A5 7E       LDA $7E          YES—Get last track number
D7CD    F0 4D       BEQ $D81C        Is number set?
D7CF    85 80       STA $80          YES—Take this as current spur
D7D1    AD 6E 02    LDA $026E        Get number of last active drive
D7D4    85 7F       STA $7F          and set as current drive
D7D6    85 E2       STA $E2          Organize drive for file
D7D8    A9 02       LDA #$02         Flag for wildcard
D7DA    85 E7       STA $E7          set
D7DC    AD 6F 02    LDA $026F        Get last sector worked with and
D7DF    85 81       STA $81          convey as curreent sector number
D7E1    20 00 C1    JSR $C100        LED on current drive goes 'on'
D7E4    20 46 DC    JSR $DC46        Open up buffer to read sector
D7E7    A9 04       LDA #$04         Bitflag for program file
D7E9    05 7F       ORA $7F          Get current drive with
D7EB[1] A6 82       LDX $82          number of current channel and
```

```
D7ED   99 EC 00   STA $00EC,Y     put filetype flag iinto channel
D7F0   4C 94 C1   JMP $C194       Prepare 'OK' message
D7F3²  E0 24      CPX #$24        Compare character with '$'
D7F5   D0 1E      BNE $D815       Should directory be loaded?
D7F7   AD 4C 02   LDA $024C       YES—Get secondary address again
D7FA   D0 03      BNE $D7FF       Load directory as a program?
D7FC   4C 55 DA   JMP $DA55       YES—Convrt directry to BASIC prg.
D7FF¹  20 D1 C1   JSR $C1D1       Set counter f/parameters in comnd
D802   AD 85 FE   LDA $FE85       Save number of directory track
D805   85 80      STA $80         as current track
D807   A9 00      LDA #$00        Set start sector of directory as
D809   85 81      STA $81         current track number
D80B   20 46 DC   JSR $DC46       Open buffer -- read  in sector
D80E   A5 7F      LDA $7F         Get current drive number
D810   09 02      ORA #$02        Set SEQ file flag & save directry
D812   4C EB D7   JMP $D7EB       as a file; end
D815¹  E0 23      CPX #$23        Compare char. with '#'
D817   D0 12      BNE $D82B       Direct access channel open?
D819   4C 84 CB   JMP $CB84       YES—Open direct access file
D81C¹  A9 02      LDA #$02        Set identifier for
D81E   8D 96 02   STA $0296       PRG file
D821   A9 00      LDA #$00        Establish drive 0 as
D823   85 7F      STA $7F         current drive
D825   8D 8E 02   STA $028E       Set pointer to last drive
D828   20 42 D0   JSR $D042       Read BAM into buffer
D82B¹  20 E5 C1   JSR $C1E5       Look for command string after ':'
D82E   D0 04      BNE $D834       Found it?
D830   A2 00      LDX #$00        YES—Startposition of parameters
D832   F0 0C      BEQ $D840       Jump to $D840
D834¹  8A         TXA             Get number of parameters
D835   F0 05      BEQ $D83C       Parameters separated by comma?
D837   A9 30      LDA #$30        YES—Display
D839   4C C8 C1   JMP $C1C8       '30 Syntax Error' error message
D83C¹  88         DEY             Set pointer to ':'
D83D   F0 01      BEQ $D840       Reached start-of-parameters?
D83F   88         DEY             Set pointer to drive assignment &
D840²  8C 7A 02   STY $027A       save position
D843   A9 8D      LDA #$8D        Look for end-of-command string
D845   20 68 C2   JSR $C268       identifier in input buffer
D848   E8         INX             Number of parameters found; save
D849   8E 78 02   STX $0278       those separated by commas
D84C   20 12 C3   JSR $C312       Get drive number; note it
D84F   20 CA C3   JSR $C3CA       Check drive number
D852   20 9D C4   JSR $C49D       Look for file entry in directory
D855   A2 00      LDX #$00        Clear pointer:
D857   8E 58 02   STX $0258       Length of a record
D85A   8E 97 02   STX $0297       File operating mode
```

```
D85D    8E 4A 02   STX $024A      Filetype
D860    E8         INX            Check  next filename
D861    EC 77 02   CPX $0277      against number of names on hand
D864    B0 10      BCS $D876      Any other tasks on hand?
D866    20 09 DA   JSR $DA09      YES-Get filetype & operating mode
D869    E8         INX            Check pointer to next filename
D86A    EC 77 02   CPX $0277      against number of names onhand
D86D    B0 07      BCS $D876      Are all names worked out?
D86F    C0 04      CPY #$04       NO-Test filetype againse REL
D871    F0 3E      BEQ $D8B1      Is there a relative file here?
D873    20 09 DA   JSR $DA09      Get filetype and operating mode
D876[4] AE 4C 02   LDX $024C      Repeat 2ndary address and set it
D879    86 83      STX $83        up; compare with start-of-
D87B    E0 02      CPX #$02       file channel
D87D    B0 12      BCS $D891      Is channel number >2?
D87F    8E 97 02   STX $0297      NO-Set read/write flag
D882    A9 40      LDA #$40       Flag for 'illegal BAM'
D884    8D F9 02   STA $02F9      set
D887    AD 4A 02   LDA $024A      Get current filetype
D88A    D0 1B      BNE $D8A7      Is there a DEL file?
D88C    A9 02      LDA #$02       YES-Set PRG file identifier
D88E    8D 4A 02   STA $024A      as current filetype
D891[1] AD 4A 02   LDA $024A      Get filetype
D894    D0 11      BNE $D8A7      Is 'DEL' type given?
D896    A5 E7      LDA $E7        YES-Get filetype frm chanel table
D898    29 07      AND #$07       Divide up and
D89A    8D 4A 02   STA $024A      save
D89D    AD 80 02   LDA $0280      Track # of sector frm buffertable
D8A0    D0 05      BNE $D8A7      Ts track set?
D8A2    A9 01      LDA #$01       NO-Set 'SEQ' identifier
D8A4    8D 4A 02   STA $024A      in current filetype
D8A7[3] AD 97 02   LDA $0297      Repeat file operation mode and
D8AA    C9 01      CMP #$01       compare with value for 'write'
D8AC    F0 18      BEQ $D8C6      Should file be written?
D8AE    4C 40 D9   JMP $D940      NO-Open read channel
D8B1[1] BC 7A 02   LDY $027A,X    Get pointer to next parameter and
D8B4    B9 00 02   LDA $0200,Y    get & store parameter characters
D8B7    8D 58 02   STA $0258      from input buffer
D8BA    AD 80 02   LDA $0280      Test fileblock track
D8BD    D0 B7      BNE $D876      Is task set?
D8BF    A9 01      LDA #$01       YES-Set read/write
D8C1    8D 97 02   STA $0297      flag
D8C4    D0 B0      BNE $D876      Jump to $D876
D8C6[1] A5 E7      LDA $E7        Get filetype
D8C8    29 80      AND #$80       Get 'wildcard onhand' flag and
D8CA    AA         TAX            save
D8CB    D0 14      BNE $D8E1      Is there a wildcard in filename?
```

```
D8CD  A9 20     LDA #$20      NO—Test 'File not closed' flag
D8CF  24 E7     BIT $E7       for first name
D8D1  F0 06     BEQ $D8D9     Has file been closed?
D8D3  20 B6 C8  JSR $C8B6     NO—Clear fileentry from directory
D8D6  4C E3 D9  JMP $D9E3     Set up for new file
D8D91 AD 80 02  LDA $0280     Track number of first file block
D8DC  D0 03     BNE $D8E1     Is file covered?
D8DE  4C E3 D9  JMP $D9E3     NO—Set up for new file
D8E12 AD 00 02  LDA $0200     Get 1st char from input buffer &
D8E4  C9 40     CMP #$40      compare w/Replace command ('@')
D8E6  F0 0D     BEQ $D8F5     Overwrite pre-existing file?
D8E8  8A        TXA           Get wildcard flag again
D8E9  D0 05     BNE $D8F0     Is file on hand?
D8EB  A9 63     LDA #$63      YES—display
D8ED  4C C8 C1  JMP $C1C8     '63 File Exists' error message
D8F01 A9 33     LDA #$33      Display
D8F2  4C C8 C1  JMP $C1C8     '33 Syntax Error' error message
----------------------------------------------------------------
[D8E6]  Overwrite corresponding file entry
D8F5  A5 E7     LDA $E7       Get filetype of 1st filename and
D8F7  29 07     AND #$07      separate flagbits
D8F9  CD 4A 02  CMP $024A     Compare w/corresponding filetype
D8FC  D0 67     BNE $D965     Identical?
D8FE  C9 04     CMP #$04      YES—Test for relative file value
D900  F0 63     BEQ $D965     Is it relative?
D902  20 DA DC  JSR $DCDA     NO—Open file for writing
D905  A5 82     LDA $82       Get # of open channel and save
D907  8D 70 02  STA $0270     as currently-open write channel
D90A  A9 11     LDA #$11      Set # for internal read channel
D90C  85 83     STA $83       (17) as secondary address
D90E  20 EB D0  JSR $D0EB     Open read channel
D911  AD 94 02  LDA $0294     Get position of current buffer
D914  20 C8 D4  JSR $D4C8     and set buffer address
D917  A0 00     LDY #$00      Initialize buffer pointer
D919  B1 94     LDA ($94),Y   Get filetype from dir. buffer
D91B  09 20     ORA #$20      Set 'file open' flagbit and
D91D  91 94     STA ($94),Y   write back into file entry
D91F  A0 1A     LDY #$1A      Set buffer pointer to position of
D921  A5 80     LDA $80       new track #; Get track number and
D923  91 94     STA ($94),Y   write in file entry
D925  C8        INY           Buffer pointer to next position
D926  A5 81     LDA $81       Get number of current sector and
D928  91 94     STA ($94),Y   save as value to be entered
D92A  AE 70 02  LDX $0270     Get current write channel number
D92D  A5 D8     LDA $D8       Get file entry sector and assign
D92F  9D 60 02  STA $0260,X   number of file entry
D932  A5 DD     LDA $DD       Get pointer to sector # in entry
```

```
D934   9D 66 02   STA $0266,X     and assign file entry
D937   20 3B DE   JSR $DE3B       Current track/sector # of job
D93A   20 64 D4   JSR $D464       Write file sector
D93D   4C EF D9   JMP $D9EF       Write data to file
D940¹  AD 80 02   LDA $0280       Get track number of first entry
D943   D0 05      BNE $D94A       Found the right entry?
D945   A9 62      LDA #$62        NO—Display
D947   4C C8 C1   JMP $C1C8       '62 File Not Found' error message
D94A¹  AD 97 02   LDA $0297       Determine file operating mode
D94D   C9 03      CMP #$03        and compare with value for 'M'
D94F   F0 0B      BEQ $D95C       Read an unclosed file?
D951   A9 20      LDA #$20        NO—Set flag for
D953   24 E7      BIT $E7         'File is not properly closed yet'
D955   F0 05      BEQ $D95C       Is flag set?
D957   A9 60      LDA #$60        YES—Display
D959   4C C8 C1   JMP $C1C8       '60 Write File Open' error messge
D95C²  A5 E7      LDA $E7         Get filetype;
D95E   29 07      AND #$07        isolate file identifier
D960   CD 4A 02   CMP $024A       Compare with current filetype
D963   F0 05      BEQ $D96A       Identical?
D965³  A9 64      LDA #$64        NO—Display
D967   4C C8 C1   JMP $C1C8       '64 Filetype Mismatch' error msge
D96A¹  A0 00      LDY #$00        Reset
D96C   8C 79 02   STY $0279       buffer pointer
D96F   AE 97 02   LDX $0297       Determine file operating mode and
D972   E0 02      CPX #$02        compare with identifier for 'A'
D974   D0 1A      BNE $D990       Should data be appended?
D976   C9 04      CMP #$04        YES—Chk filetype against REL file
D978   F0 EB      BEQ $D965       Is it a relative file?
D97A   B1 94      LDA ($94),Y     NO—Get filetype frm direc buffer
D97C   29 4F      AND #$4F        and save file open
D97E   91 94      STA ($94),Y     Put filetype back into entry
D980   A5 83      LDA $83         Get current secondary address;
D982   48         PHA             hang onto it
D983   A9 11      LDA #$11        Set internal read channel #(17)
D985   85 83      STA $83         as current secondary address
D987   20 3B DE   JSR $DE3B       Determine current track/sector
D98A   20 64 D4   JSR $D464       Write sector to diskette
D98D   68         PLA             Repeat secondary address and
D98E   85 83      STA $83         reset
D990¹  20 A0 D9   JSR $D9A0       Open file for reading
D993   AD 97 02   LDA $0297       Determine file operation,compare
D996   C9 02      CMP #$02        with identifier for 'A' (Append)
D998   D0 55      BNE $D9EF       Connect data to preexisting file?
D99A   20 2A DA   JSR $DA2A       YES—Proceed with append and
D99D   4C 94 C1   JMP $C194       get 'Ok' message ready
-------------------------------------------------------------------
```

```
[CA26/D990]
Open file for reading
D9A0  A0 13     LDY #$13       Turn pointer to side-sector entry
D9A2  B1 94     LDA ($94),Y    Get track # of 1st side-sector &
D9A4  8D 59 02  STA $0259      save itken
D9A7  C8        INY            Buffer pointer to next byte
D9A8  B1 94     LDA ($94),Y    Get sector # of first side-sector
D9AA  8D 5A 02  STA $025A      and save it
D9AD  C8        INY            Buffer pointer to next position
D9AE  B1 94     LDA ($94),Y    Determine length of a record
D9B0  AE 58 02  LDX $0258      Get last record length
D9B3  8D 58 02  STA $0258      Set new record length
D9B6  8A        TXA            Last record length
D9B7  F0 0A     BEQ $D9C3      not set?
D9B9  CD 58 02  CMP $0258      NO—Compare with current length
D9BC  F0 05     BEQ $D9C3      Reached the last record?
D9BE  A9 50     LDA #$50       YES—Display
D9C0  20 C8 C1  JSR $C1C8      '50Record Not Present' err.messge
D9C3² AE 79 02  LDX $0279      Number of filenames (0)
D9C6  BD 80 02  LDA $0280,X    Get current track number and
D9C9  85 80     STA $80        set
D9CB  BD 85 02  LDA $0285,X    Get current sector number and
D9CE  85 81     STA $81        set it
D9D0  20 46 DC  JSR $DC46      Open read channel
D9D3  A4 82     LDY $82        Get channel number
D905  AE 79 02  LDX $0279      Number of files worked on
D9D8  B5 D8     LDA $D8,X      Get sector number and
D9DA  99 60 02  STA $0260,Y    transfer
D9DD  B5 DD     LDA $DD,X      Get position in file entry and
D9DF  99 66 02  STA $0266,Y    transfer
D9E2  60        RTS            Return from this subroutine
-------------------------------------------------------------
```

```
[D8D6/D8DE]
Open file for writing
D9E3   A5 E2       LDA $E2        Establish number of disk drive
D9E5   29 01       AND #$01       to be utilized
D9E7   85 7F       STA $7F        and save as current drive
D9E9   20 DA DC    JSR $DCDA      Open channel for reading
D9EC   20 E4 D6    JSR $D6E4      Enter file in directory
D9EF2  A5 83       LDA $83        Get current 2ndary address & test
D9F1   C9 02       CMP #$02       for start-of-data channel
D9F3   B0 11       BCS $DA06      Should a LOAD or SAVE be done?
D9F5   20 3E DE    JSR $DE3E      YES—Get track/sector from job
D9F8   A5 80       LDA $80        Save track # of last track of
D9FA   85 7E       STA $7E        last access
D9FC   A5 7F       LDA $7F        Get current drive number and
D9FE   8D 6E 02    STA $026E      save as last active drive
DA01   A5 81       LDA $81        Save current sector as the last
DA03   8D 6F 02    STA $026F      accessed
DA061  4C 99 C1    JMP $C199      Prepare 'OK' message
------------------------------------------------------------------
[D866/D873]
Set up filetype and file operation as command string
DA09   BC 7A 02    LDY $027A,X    Get position of first parameter
DA0C   B9 00 02    LDA $0200,Y    Get character from input buffer
DA0F   A0 04       LDY #$04       Set counter to # operating modes
DA111  88          DEY            Turn counter to next identifier
DA12   30 08       BMI $DA1C      All operating modes checked?
DA14   D9 B2 FE    CMP $FEB2,Y    NO—Compare w/file operations mode
DA17   D0 F8       BNE $DA11      Identical?
DA19   8C 97 02    STY $0297      YES—Save position in input string
DA1C1  A0 05       LDY #$05       and set counter for filetype
DA1E1  88          DEY            Turn counter to next filetype
DA1F   30 08       BMI $DA29      All filetypes already checked?
DA21   D9 B6 FE    CMP $FEB6,Y    NO—Compare with filetype
DA24   D0 F8       BNE $DA1E      Identical?
DA26   8C 4A 02    STY $024A      YES—Save position
DA291  60          RTS            Return from this subroutine
------------------------------------------------------------------
[C996/D99A/DA32]
Prepare file for Append
DA2A   20 39 CA    JSR $CA39      Read data byte
DA2D   A9 80       LDA #$80       Test flag for
DA2F   20 A6 DD    JSR $DDA6      'EOI reached'
DA32   F0 F6       BEQ $DA2A      Has last byte been read?
DA34   20 95 DE    JSR $DE95      YES—Find current track & sector
DA37   A6 81       LDX $81        Get pointer to correct data reg.
DA39   E8          INX            and increment by 1
```

```
DA3A   8A          TXA             (if $FF, then make it 0 again)
DA3B   D0 05       BNE $DA42       Is the sector completely filled?
DA3D   20 A3 D1    JSR $D1A3       YES—Write sector to diskette
DA40   A9 02       LDA #$02        Set buffer pointer to beginning
DA42¹  20 C8 D4    JSR $D4C8       of data register
DA45   A6 82       LDX $82         Get current channel number
DA47   A9 01       LDA #$01        Set write flag in
DA49   95 F2       STA $F2,X       channel status table
DA4B   A9 80       LDA #$80        Combine write flag
DA4D   05 82       ORA $82         with channel number
DA4F   A6 83       LDX $83         Get current secondary address and
DA51   9D 2B 02    STA $022B,X     assign status byte of 2ndary adr.
DA54   60          RTS             Return from this subroutine
-------------------------------------------------------------------
[D7FC]
Transmit directory to computer
DA55   A9 0C       LDA #$0C        Set command
DA57   8D 2A 02    STA $022A       number 12
DA5A   A9 00       LDA #$00        Drive number
DA5C   AE 74 02    LDX $0274       Get length of command string and
DA5F   CA          DEX             compare with 1
DA60   F0 0B       BEQ $DA6D       Does command only have one char.?
DA62   CA          DEX             NO—Compare to 2
DA63   D0 21       BNE $DA86       Does command have only 2 chars?
DA65   AD 01 02    LDA $0201       YES—Get 2nd character and check
DA68   20 BD C3    JSR $C3BD       for drive number
DA6B   30 19       BMI $DA86       Is drive assignment error-free?
DA6D¹  85 E2       STA $E2         YES—Save drive
DA6F   EE 77 02    INC $0277       Pointer to first
DA72   EE 78 02    INC $0278       and second parameter in command
DA75   EE 7A 02    INC $027A       string--move to next position
DA78   A9 80       LDA #$80        Set flag for 'file properly
DA7A   85 E7       STA $E7         closed'
DA7C   A9 2A       LDA #$2A        Set '*' wildcard as filename in
DA7E   8D 00 02    STA $0200       command
DA81   8D 01 02    STA $0201       string
DA84   D0 18       BNE $DA9E       Jump to $DA9E
DA86²  20 E5 C1    JSR $C1E5       Seek ':' in command string
DA89   D0 05       BNE $DA90       Found a colon?
DA8B   20 DC C2    JSR $C2DC       YES—Clear pointr f/command string
DA8E   A0 03       LDY #$03        Position of first filename (1)
DA90¹  88          DEY             Set pointer to position
DA91   88          DEY             of name
DA92   8C 7A 02    STY $027A       Set pointer to first filename
DA95   20 00 C2    JSR $C200       Set pointer f/parameter analysis
DA98   20 98 C3    JSR $C398       Pointer to filename and filetype
```

```
DA9B   20 20 C3   JSR $C320     Get drive # from command string
DA9E1  20 CA C3   JSR $C3CA     Prep drive for access
DAA1   20 B7 C7   JSR $C7B7     Produce directory title
DAA4   20 9D C4   JSR $C49D     Get filename from directory
DAA7   20 9E EC   JSR $EC9E     Ascertain directory line
DAAA   20 37 D1   JSR $D137     1st byte of directory from buffer
DAAD   A6 82      LDX $82       Current channel number
DAAF   9D 3E 02   STA $023E,X   Prep byte for output
DAB2   A5 7F      LDA $7F       Save current drive as drive used
DAB4   8D 8E 02   STA $028E     for last access
DAB7   09 04      ORA #$04      Set flag for PRG file
DAB9   95 EC      STA $EC,X     and put in channel table
DABB   A9 00      LDA #$00      Get back pointer in input
DABD   85 A3      STA $A3       buffer
DABF   60         RTS           Return from this subroutine
----------------------------------------------------------------------
[8151/9193/91A3/91B9/A9F1/BF60/E8CE]
Close file
DAC0   A9 00      LDA #$00      Set 'illegal BAM'
DAC2   8D F9 02   STA $02F9     flag
DAC5   A5 83      LDA $83       Get current secondary address
DAC7   D0 0B      BNE $DAD4     LOAD command?
DAC9   A9 00      LDA #$00      YES—Clear 'Directory will be
DACB   8D 54 02   STA $0254     displayed' flag
DACE   20 27 D2   JSR $D227     Close channel
DAD11  4C DA D4   JMP $D4DA     Close internal read/write chanels
DAD41  C9 0F      CMP #$0F      Compare secondary address w/ 15
DAD6   F0 14      BEQ $DAEC     Command channel addresssed?
DAD8   20 02 DB   JSR $DB02     NO—Close file
DADB   A5 83      LDA $83       Get current  secondary address
DADD   C9 02      CMP #$02      Compare with begin.of filechannel
DADF   90 F0      BCC $DAD1     Channel have LOAD/SAVE (0/1)?
DAE1   AD 6C 02   LDA $026C     NO—Get error flag and test it
DAE4   D0 03      BNE $DAE9     Run into an error?
DAE6   4C 94 C1   JMP $C194     NO—'Ok' message displayed
DAE91  4C AD C1   JMP $C1AD     Display error message
----------------------------------------------------------------------
```

```
[DAD6]
Close all files
DAEC   A9 0E      LDA #$0E        Highest 2ndary address for  files
DAEE   85 83      STA $83         set as current secondary address
DAF0¹  20 02 DB   JSR $DB02       Close file
DAF3   C6 83      DEC $83         Go to next secondary address
DAF5   10 F9      BPL $DAF0       All channels closed?
DAF7   AD 6C 02   LDA $026C       YES—Get error flag and test
DAFA   D0 03      BNE $DAFF       Closures done without errors?
DAFC   4C 94 C1   JMP $C194       YES—Display 'OK' message
DAFF¹  4C AD C1   JMP $C1AD       Display error message
-------------------------------------------------------------
[C9F7/DADB/DAF0]
Files declared through secondary address, closed
DB02   A6 83      LDX $83         Get current secondary address and
DB04   BD 2B 02   LDA $022B,X     determine corresponding status
DB07   C9 FF      CMP #$FF        Compare with 'Free channel' value
DB09   D0 01      BNE $DB0C       Is channel out?
DB0B   60         RTS             NO—Return from this subroutine
DB0C¹  29 0F      AND #$0F        Determine clear channel number
DB0E   85 82      STA $82         and save it
DB10   20 25 D1   JSR $D125       Get filetype & compare w/directry
DB13   C9 07      CMP #$07        access identifier
DB15   F0 0F      BEQ $DB26       Identical?
DB17   C9 04      CMP #$04        NO—Check with value for REL file
DB19   F0 11      BEQ $DB2C       Identical?
DB1B   20 07 D1   JSR $D107       NO—Check channel f/write channel
DB1E   B0 09      BCS $DB29       Is the write channel open?
DB20   20 62 DB   JSR $DB62       YES—Write to end
DB23   20 A5 DB   JSR $DBA5       Close directory entry
DB26¹  20 F4 EE   JSR $EEF4       Write BAM back to diskette
DB29¹  4C 27 D2   JMP $D227       Close channel
DB2C¹  20 F1 DD   JSR $DDF1       Write current buffer to diskette
DB2F   20 1E CF   JSR $CF1E       Apple new buffer
DB32   20 CB E1   JSR $E1CB       Get position of last record
DB35   A6 D5      LDX $D5         Get number of last side-sector
DB37   86 73      STX $73         and save
DB39   E6 73      INC $73         Choose  next side-sector
DB3B   A9 00      LDA #$00        Direct zeropage addresses as
DB3D   85 70      STA $70         temporary
DB3F   85 71      STA $71         storage
DB41   A5 D6      LDA $D6         Get position of side-sector
DB43   38         SEC             consider number of bytes
DB44   E9 0E      SBC #$0E        for chaining of
DB46   85 72      STA $72         other side-sectors
DB48   20 51 DF   JSR $DF51       Calculate number of file blocks
```

```
DB4B   A6 82      LDX $82        Current channel number
DB4D   A5 70      LDA $70        Put # of relative file  blocks
DB4F   95 B5      STA $B5,X      (low-byte) in Table
DB51   A5 71      LDA $71        Copy
DB53   95 BB      STA $BB,X      high-byte
DB55   A9 40      LDA #$40       Check filetype channel flag
DB57   20 A6 DD   JSR $DDA6      for 'entry correct'
DB5A   F0 03      BEQ $DB5F      Is flag in filetype set?
DB5C   20 A5 DB   JSR $DBA5      YES—Realize directoy entry
DB5F1  4C 27 D2   JMP $D227      Close channel
-------------------------------------------------------------------
[DB20]
Write last sector of  a file to diskette
DB62   A6 82      LDX $82        Get current channel number
DB64   B5 B5      LDA $B5,X      Get channel-arranged
DB66   15 BB      ORA $BB,X      record number and test it
DB68   D0 0C      BNE $DB76      Is record number set?
DB6A   20 E8 D4   JSR $D4E8      NO—Get current buffer pointer and
DB6D   C9 02      CMP #$02       compare with start of filerange
DB6F   D0 05      BNE $DB76      Is the sector still empty?
DB71   A9 0D      LDA #$0D       YES—Write empty record (<RETURN>)
DB73   20 F1 CF   JSR $CFF1      to buffer
DB762  20 E8 D4   JSR $D4E8      Get buffer pointer; compare with
DB79   C9 02      CMP #$02       start-of-filerange
DB7B   D0 0F      BNE $DB8C      Is the sector still empty?
DB7D   20 1E CF   JSR $CF1E      YES—Open new buffer
DB80   A6 82      LDX $82        Determine current channel number
DB82   B5 B5      LDA $B5,X      Predetermined record # (lo-byte)
DB84   D0 02      BNE $DB88      Is low-byte = zero?
DB86   D6 BB      DEC $BB,X      YES—Decrement hi-byt/record no.-1
DB881  D6 B5      DEC $B5,X      Decrement low-byte by 1
DB8A   A9 00      LDA #$00       Value for 'buffer full'
DB8C1  38         SEC            Calculate number of applicable
DB8D   E9 01      SBC #$01       filebytes per sector
DB8F   48         PHA            and note it
DB90   A9 00      LDA #$00       Set buffer pointer for
DB92   20 C8 D4   JSR $D4C8      connected bytes
DB95   20 F1 CF   JSR $CFF1      Write identifier for last sector
DB98   68         PLA            Get # of applicable filebytes
DB99   20 F1 CF   JSR $CFF1      and write to sector
DB9C   20 C7 D0   JSR $D0C7      Write sector back to diskette and
DB9F   20 99 D5   JSR $D599      wait until job is completed
DBA2   4C 1E CF   JMP $CF1E      Open new buffer
-------------------------------------------------------------------
```

```
[DB23/DB5C]
Close directory entry after write operation
DBA5  A6 82     LDX $82        Get current channel number and
DBA7  8E 70 02  STX $0270      retain it
DBAA  A5 83     LDA $83        Get # of current 2ndary address
DBAC  48        PHA            and retain it
DBAD  BD 60 02  LDA $0260,X    Get # of directry sector f/entry
DBB0  85 81     STA $81        and set as current sector
DBB2  BD 66 02  LDA $0266,X    Get entry positiion in directory
DBB5  8D 94 02  STA $0294      and set as current buffer pointer
DBB8  B5 EC     LDA $EC,X      Get filetype of channel
DBBA  29 01     AND #$01       Determine drive number and
DBBC  85 7F     STA $7F        take up as current drive
DBBE  AD 85 FE  LDA $FE85      Get number of directory track
DBC1  85 80     STA $80        and set up as current track
DBC3  20 93 DF  JSR $DF93      Get and save
DBC6  48        PHA            buffer number
DBC7  85 F9     STA $F9        Set current buffer number
DBC9  20 60 D4  JSR $D460      Read directory sector into buffer
DBCC  A0 00     LDY #$00       Reset position pointer
DBCE  BD E0 FE  LDA $FEE0,X    Get buffr address(hi-byte), take
DBD1  85 87     STA $87        as high-byte of buffer pointer
DBD3  AD 94 02  LDA $0294      Get current position in buffer &
DBD6  85 86     STA $86        set as low-byte
DBD8  B1 86     LDA ($86),Y    Get filetype frm directry entry &
DBDA  29 20     AND #$20       check for 'file open' flag
DBDC  F0 43     BEQ $DC21      File already closed?
DBDE  20 25 D1  JSR $D125      NO—Test filetype further and test
DBE1  C9 04     CMP #$04       against value for relative file
DBE3  F0 44     BEQ $DC29      Identical?
DBE5  B1 86     LDA ($86),Y    NO—Get entire filetype pointer
DBE7  29 8F     AND #$8F       Clear flags
DBE9  91 86     STA ($86),Y    and filetype again back to entry
DBEB  C8        INY            Buffer pointer to next position
DBEC  B1 86     LDA ($86),Y    Get track # of first sector/file
DBEE  85 80     STA $80        and save as current track
DBF0  84 71     STY $71        Save current buffer pointer
DBF2  A0 1B     LDY #$1B       Set buffer pointer of sector from
DBF4  B1 86     LDA ($86),Y    overwrite and get number
DBF6  48        PHA            Save sector number
DBF7  88        DEY            Set buffer pointer to appropriate
DBF8  B1 86     LDA ($86),Y    track and get track number
DBFA  D0 0A     BNE $DC06      No overwrite entry set?
DBFC  85 80     STA $80        YES—Get track & sector number
DBFE  68        PLA            again, and put into
DBFF  85 81     STA $81        current pointer
```

```
DC01   A9 67       LDA #$67        Display '67 Illegal Track
DC03   20 45 E6    JSR $E645       Or Sector' error message
DC06¹  48          PHA             Save track number
DC07   A9 00       LDA #$00        Clear track and
DC09   91 86       STA ($86),Y     sector number
DC0B   C8          INY             of the file entry to be
DC0C   91 86       STA ($86),Y     overwritten
DC0E   68          PLA             Get track number again
DC0F   A4 71       LDY $71         Reset buffer pointer
DC11   91 86       STA ($86),Y     Set track to first sector of file
DC13   C8          INY             Buffer pointer to next byte
DC14   B1 86       LDA ($86),Y     Get number of old sector and
DC16   85 81       STA $81         save it
DC18   68          PLA             Get # of first sector of file
DC19   91 86       STA ($86),Y     and store in entry
DC1B   20 7D C8    JSR $C87D       Clear old file sectors
DC1E   4C 29 DC    JMP $DC29       Close file
DC21¹  B1 86       LDA ($86),Y     Get filetype from entry
DC23   29 0F       AND #$0F        Isolate file identifiers
DC25   09 80       ORA #$80        Set 'file closed' flag and set up
DC27   91 86       STA ($86),Y     as new filetype
DC29²  AE 70 02    LDX $0270       Repeat number of current channel
DC2C   A0 1C       LDY #$1C        Set buff pntr to block assign(28)
DC2E   B5 B5       LDA $B5,X       Get # of blocks to a file(lobyte)
DC30   91 86       STA ($86),Y     and write to entry
DC32   C8          INY             Set buffer pointer to next byte
DC33   B5 BB       LDA $BB,X       Get hi-byte of block # and write
DC35   91 86       STA ($86),Y     to entry
DC37   68          PLA             Recall current buffer number --
DC38   AA          TAX             note it
DC39   A9 90       LDA #$90        Jobcode for 'write sector'
DC3B   05 7F       ORA $7F         Enter current drive in jobcode
DC3D   20 90 D5    JSR $D590       Execute job
DC40   68          PLA             Repeat and reset current
DC41   85 83       STA $83         secondary address
DC43   4C 07 D1    JMP $D107       Get channel number
--------------------------------------------------------------------------
[D47E/D7E4/D80B/D9D0/DC98:DD8A]
Open channel to read a file
DC46   A9 01       LDA #$01        Buffer number
DC48   20 E2 D1    JSR $D1E2       Open channel for reading
DC4B   20 B6 DC    JSR $DCB6       Set channel pointer
DC4E   AD 4A 02    LDA $024A       Get current filetype and
DC51   48          PHA             note it
DC52   0A          ASL A           Establish filetype entry for
DC53   05 7F       ORA $7F         chanel table;concentrate on drive
```

```
DC55   95 EC       STA $EC,X       and assign to channel table
DC57   20 9B D0    JSR $D09B       Read sector in buffer
DC5A   A6 82       LDX $82         Get current channel number
DC5C   A5 80       LDA $80         Determine current track number
DC5E   D0 05       BNE $DC65       Are there other sectors onhand?
DC60   A5 81       LDA $81         NO-Get number of applicable
DC62   9D 44 02    STA $0244,X     filebytes and save them
DC65¹  68          PLA             Call back filetype & compare with
DC66   C9 04       CMP #$04        value for  relative files
DC68   D0 3F       BNE $DCA9       Is it a relative file?
DC6A   A4 83       LDY $83         YES-Get present secondary address
DC6C   B9 2B 02    LDA $022B,Y     Get preset channel number
DC6F   09 40       ORA #$40        Read flag
DC71   99 2B 02    STA $022B,Y     Assign secondary address
DC74   AD 58 02    LDA $0258       Get record length and
DC77   95 C7       STA $C7,X       assign to channel
DC79   20 8E D2    JSR $D28E       Open new buffer for side-sector
DC7C   10 03       BPL $DC81       Still a free buffer?
DC7E   4C 0F D2    JMP $D20F       NO-Display'70 No Channel' error
DC81¹  A6 82       LDX $82         Number of current channel
DC83   95 CD       STA $CD,X       Save buffer number
DC85   AC 59 02    LDY $0259       Get track # of 1st side-sector &
DC88   84 80       STY $80         set as current track number
DC8A   AC 5A 02    LDY $025A       Get sector number and store as
DC8D   84 81       STY $81         current sector
DC8F   20 D3 D6    JSR $D6D3       Track and sector in job loop
DC92   20 73 DE    JSR $DE73       Read sidesector frm disk to buffr
DC95   20 99 D5    JSR $D599       Wait until job is run
DC98¹  A6 82       LDX $82         Get current channel number and
DC9A   A9 02       LDA #$02        initialize record pointer
DC9C   95 C1       STA $C1,X       in table
DC9E   A9 00       LDA #$00        Pointer to start-of-sector
DCA0   20 C8 D4    JSR $D4C8       Set buffer pointer
DCA3   20 53 E1    JSR $E153       Read first record and prepare it
DCA6   4C 3E DE    JMP $DE3E       Set byte for output  and pointer
DCA9¹  20 56 D1    JSR $D156       Get byte from buffer
DCAC   A6 82       LDX $82         Get current channel number
DCAE   9D 3E 02    STA $023E,X     Send byte for output
DCB1   A9 88       LDA #$88        Set output flag in table
DCB3   95 F2       STA $F2,X       with channel status
DCB5   60          RTS             Return from this subroutine
-------------------------------------------------------------
```

```
[DC4B/DCE5]
Initialize 'channel open' pointer
DCB6   A6 82      LDX $82        Get channel number and determine
DCB8   B5 A7      LDA $A7,X      preset buffer
DCBA   0A         ASL A          Test 'buffer open' flag
DCBB   30 06      BMI $DCC3      Buffer covered?
DCBD   A8         TAY            NO--
DCBE   A9 02      LDA #$02       Set buffer pointer to start of
DCC0   99 99 00   STA $0099,Y    file range (byte $02)
DCC3¹  B5 AE      LDA $AE,X      Get buffer status and
DCC5   09 80      ORA #$80       set 'buffer in-
DCC7   95 AE      STA $AE,X      active' flag
DCC9   0A         ASL A          Test bit 6
DCCA   30 06      BMI $DCD2      Should buffer be written back?
DCCC   A8         TAY            NO—Save buffer number
DCCD   A9 02      LDA #$02       Set buffer pointer (low-byte) to
DCCF   99 99 00   STA $0099,Y    start-of-filerange
DCD2¹  A9 00      LDA #$00       Clear number of blocks free
DCD4   95 B5      STA $B5,X      (low-byte)
DCD6   4C 7F A9   JMP $A97F      Clear number of blocks free
DCD9   EA         NOP            unused
-------------------------------------------------------------------
[D48A/D902/D9E9]
Open channel to write to file
DCDA   20 A9 F1   JSR $F1A9      Look for free sector in BAM
DCDD   A9 01      LDA #$01       Number of buffers needed
DCDF   20 DF D1   JSR $D1DF      Cover buffer
DCE2   20 D0 D6   JSR $D6D0      Trck and sector to job loop
DCE5   20 B6 DC   JSR $DCB6      Initialize channel pointer
DCE8   A6 82      LDX $82        Number of present channel
DCEA   AD 4A 02   LDA $024A      Get current filetype
DCED   48         PHA            and save
DCEE   0A         ASL A          Take drive number into
DCEF   05 7F      ORA $7F        filetype and assign
DCF1   95 EC      STA $EC,X      to table of that channel
DCF3   68         PLA            Get original filetype and check
DCF4   C9 04      CMP #$04       for value for relative file
DCF6   F0 05      BEQ $DCFD      Is there a relative file?
DCF8   A9 01      LDA #$01       NO—Set write
DCFA   95 F2      STA $F2,X      flag
DCFC   60         RTS            Return from this subroutine
DCFD¹  A4 83      LDY $83        Get current secondary address
DCFF   B9 2B 02   LDA $022B,Y    and assign channel
DD02   29 3F      AND #$3F       Reset flag bits in channel status
DD04   09 40      ORA #$40       Set read
DD06   99 2B 02   STA $022B,Y    flag
```

```
DD09   AD 58 02   LDA $0258    Get length of a record and
DD0C   95 C7      STA $C7,X    save it
DD0E   20 8E D2   JSR $D28E    Look for a buffer
DD11   10 03      BPL $DD16    Found a free buffer?
DD13   4C 0F D2   JMP $D20F    NO—Display '70 No Channel' error
DD16¹  A6 82      LDX $82      Get current channel number and
DD18   95 CD      STA $CD,X    connect buffer for side-sector
DD1A   20 C1 DE   JSR $DEC1    Clear buffer contents
DD1D   20 1E F1   JSR $F11E    Look for free BAM sector
DD20   A5 80      LDA $80      Store track # of the sector as
DD22   8D 59 02   STA $0259    the track for first side-sector
DD25   A5 81      LDA $81      Number of sector marked as sector
DD27   8D 5A 02   STA $025A    number for first side-sector
DD2A   A6 82      LDX $82      Get current channel number
DD2C   B5 CD      LDA $CD,X    Get # of corresponding buffer
DD2E   20 D3 D6   JSR $D6D3    Track & sector #'s to job loop
DD31   A9 00      LDA #$00     Set buffer pointer to start-
DD33   20 E9 DE   JSR $DEE9    of-buffer
DD36   A9 00      LDA #$00     Write identifier f/last sector to
DD38   20 8D DD   JSR $DD8D    buffer
DD3B   A9 11      LDA #$11     Put # of applicable side-sector
DD3D   20 8D DD   JSR $DD8D    bytes in buffer (17)
DD40   A9 00      LDA #$00     Transfer number of side-sector
DD42   20 8D DD   JSR $DD8D    to buffer
DD45   AD 58 02   LDA $0258    Enter record length
DD48   20 8D DD   JSR $DD8D    in side-sector
DD4B   A5 80      LDA $80      Store current track number
DD4D   20 8D DD   JSR $DD8D    in side-sector
DD50   A5 81      LDA $81      Store current sector number
DD52   20 8D DD   JSR $DD8D    in side-sector
DD55   A9 10      LDA #$10     Set buffer pointer to record
DD57   20 E9 DE   JSR $DEE9    data in side-sector
DD5A   20 3E DE   JSR $DE3E    Get track & sector of last job
DD5D   A5 80      LDA $80      Write track # of 1st file sector
DD5F   20 8D DD   JSR $DD8D    in side-sector
DD62   A5 81      LDA $81      Take number of first file sector
DD64   20 8D DD   JSR $DD8D    into side-sector
DD67   20 6C DE   JSR $DE6C    Write side-sector to diskette
DD6A   20 99 D5   JSR $D599    Wait until job is run
DD6D   A9 02      LDA #$02     Set current buffer pointer to
DD6F   20 C8 D4   JSR $D4C8    start of filerange
DD72   A6 82      LDX $82      Get number of current
DD74   38         SEC          channel
DD75   A9 00      LDA #$00     Initialize accumulator and
DD77   F5 C7      SBC $C7,X    calc. & set pos. of next record
DD79   95 C1      STA $C1,X    from record length
```

```
DD7B  20 E2 E2   JSR $E2E2        Apply record to sector
DD7E  20 19 DE   JSR $DE19        Set chaining
DD81  20 5E DE   JSR $DE5E        Write sector to diskette; wait
DD84  20 99 D5   JSR $D599        intil job is run
DD87  20 F4 EE   JSR $EEF4        Write new BAM to diskette
DD8A  4C 98 DC   JMP $DC98        Display return message
-------------------------------------------------------------------
[DD38/DD3D/DD42/DD48/DD4D/DD52/DD5F/DD64/E3FA/E3FE]
Write a byte to current side-sector
DD8D  48         PHA              Save byte
DD8E  A6 82      LDX $82          Get current channel #, &determine
DD90  B5 CD      LDA $CD,X        corresponding buffer
DD92  4C FD CF   JMP $CFFD        Transfer byte in buffer
-------------------------------------------------------------------
[Original jump is not used in DOS]
Channel number in filetype flag set (Carry=1) or cleared (Carry=0)
DD95  90 06      BCC $DD9D        Flags cleared?
-------------------------------------------------------------------
[CA4F/DD97/E01A/E0A0/E107/E25F]
Value combined in filetype (Bit =1: set)
DD97  A6 82      LDX $82          NO—Get number of current channel&
DD99  15 EC      ORA $EC,X        put into filetype flag
DD9B  D0 06      BNE $DDA3        Jump to $DDA3
-------------------------------------------------------------------
[CFAC/DD95/DFD2/E003/E0ED/E21B]
Remove value from filetype flag (Bit =1: taken out)
DD9D  A6 82      LDX $82          Get current channel number
DD9F  49 FF      EOR #$FF         and invert it
DDA1  35 EC      AND $EC,X        Mask filetype flag number
DDA3  95 EC      STA $EC,X        Set new filetype
DDA5  60         RTS              Return from this subroutine
-------------------------------------------------------------------
[C9DD/DA2F/DB57/DFD7/E0AD/E0BE/E0F5/E122/E26A]
Check for set filetype flag (flag-value in accumulator)
DDA6  A6 82      LDX $82          Get current channel number and
DDA8  35 EC      AND $EC,X        test corresponding flag
DDAA  60         RTS              Return from this subroutine
-------------------------------------------------------------------
[CF4C/E052/E060] Test whether jobcode is set up for writing
DDAB  20 93 DF   JSR $DF93        Get number of present buffer and
DDAE  AA         TAX              save it
DDAF  BD 5B 02   LDA $025B,X      Get last jobcode from buffer and
DDB2  29 F0      AND #$F0         and prepare command bits
DDB4  C9 90      CMP #$90         Compare with write value
DDB6  60         RTS              Return from this subroutine
-------------------------------------------------------------------
```

```
[C835]
Test file pointer
DDB7  A2 00      LDX #$00      Set secondary address
DDB91 86 71      STX $71       and note it
DDBB  BD 2B 02   LDA $022B,X   Get matching channel number
DDBE  C9 FF      CMP #$FF      Compare with "channel free" value
DDC0  D0 08      BNE $DDCA     Is channel covered?
DDC23 A6 71      LDX $71       YES-Repeat 2ndary address number&
DDC4  E8         INX           choose next address
DDC5  E0 10      CPX #$10      Compare with maximum address +1
DDC7  90 F0      BCC $DDB9     Is 2ndary addrs in allowed range?
DDC9  60         RTS           NO-Return from this subroutine
DDCA1 86 71      STX $71       Save free secondary address
DDCC  29 3F      AND #$3F      Determine channel number
DDCE  A8         TAY           and note it
DDCF  B9 EC 00   LDA $00EC,Y   Get filetype flag and chosen
DDD2  29 01      AND #$01      disk drive number
DDD4  85 70      STA $70       Save drive number
DDD6  AE 53 02   LDX $0253     Entry number
DDD9  B5 E2      LDA $E2,X     Get standard drive's
DDDB  29 01      AND #$01      channel and compare with
DDDD  C5 70      CMP $70       drive chosen
DDDF  D0 E1      BNE $DDC2     Identical?
DDE1  B9 60 02   LDA $0260,Y   YES-Get directory sector number
DDE4  D5 D8      CMP $D8,X     and compare with sector of entry
DDE6  D0 DA      BNE $DDC2     Identical?
DDE8  B9 66 02   LDA $0266,Y   YES-Get position of entry and
DDEB  D5 DD      CMP $DD,X     test for position in directory
DDED  D0 D3      BNE $DDC2     Identical?
DDEF  18         CLC           YES-Flag for all pointers OK
DDF0  60         RTS           Return from this subroutine
------------------------------------------------------------------
```

```
[DB2C/E2AA/E454]
Write buffer to diskette
DDF1   20 9E DF    JSR $DF9E        Test buffer status
DDF4   50 06       BVC $DDFC        Has data in buffer been changed?
DDF6   20 5E DE    JSR $DE5E        YES—write sector to diskette
DDF9   20 99 D5    JSR $D599        Wait until job is executed
DDFC1  60          RTS              Return from this subroutine
-------------------------------------------------------------------
[E3AC/E3BF]
Set chained bytes which point to next sector
DDFD   20 2B DE    JSR $DE2B        Set current buffer pointer
DE00   A5 80       LDA $80          Transfer track # of next sector
DE02   91 94       STA ($94),Y      to buffer
DE04   C8          INY              Buffer pointer to next position
DE05   A5 81       LDA $81          Write number of next sector to
DE07   91 94       STA ($94),Y      current buffer
DE09   4C 05 E1    JMP $E105        Buffer marked as 'changed'
-------------------------------------------------------------------
[E2AD/E3D7]
Get linked bytes, which point to the next sector
DE0C   20 2B DE    JSR $DE2B        Set current buffer pointer
DE0F   B1 94       LDA ($94),Y      Get track # of next sector from
DE11   85 80       STA $80          buffer and save it down
DE13   C8          INY              Set buffer pointer to next byte
DE14   B1 94       LDA ($94),Y      Get # of next sector from buffer;
DE16   85 81       STA $81          save as current sector
DE18   60          RTS              Return from this subroutine
-------------------------------------------------------------------
[DD7E/E3D1]
Set indicator for last sector in linked bytes
DE19   20 2B DE    JSR $DE2B        Set current buffer pointer
DE1C   A9 00       LDA #$00         Write identifier for last sector
DE1E   91 94       STA ($94),Y      to the buffer
DE20   C8          INY              Buffer pointer to next byte
DE21   A6 82       LDX $82          Get current channel number
DE23   B5 C1       LDA $C1,X        Get # of applicable file bytes
DE25   AA          TAX              from table and
DE26   CA          DEX              correct it
DE27   8A          TXA              (including 0)
DE28   91 94       STA ($94),Y      Write number to buffer
DE2A   60          RTS              Return from this subroutine
-------------------------------------------------------------------
```

```
[DDFD/DE0C/DE19/E1B2/E2E2] Set current buffer's pointer to zero
DE2B   20 93 DF    JSR $DF93       Get buffer number and
DE2E   0A          ASL A           double it (pointer table works
DE2F   AA          TAX             with 2-byte values)
DE30   B5 9A       LDA $9A,X       Get hi-byte of buffer address and
DE32   85 95       STA $95         send to buffer pointer
DE34   A9 00       LDA #$00        Set low-byte to start-of-
DE36   85 94       STA $94         buffer
DE38   A0 00       LDY #$00        Reset index pointer to beginning
DE3A   60          RTS             Return from this subroutine
-----------------------------------------------------------------
[C5E8/C634/D48D/D6F4/D937/D987]
Get current track and sector of current job
DE3B   20 EB D0    JSR $D0EB       Get channel # of 2ndary address
-----------------------------------------------------------------
[D9F5/DCA6/DD5A/E2D0/E3E0/E824/E840/F11E]
Get track and sector of current job
DE3E   20 93 DF    JSR $DF93       Determine corresponding buffer #
DE41   85 F9       STA $F9         and save as current buffer
DE43   0A          ASL A           Double buffer # (track/sector
DE44   A8          TAY             table works with 2-byte values)
DE45   B9 06 00    LDA $0006,Y     Get track number of current job &
DE48   85 80       STA $80         save as current track number
DE4A   B9 07 00    LDA $0007,Y     Get concurrent sector number &
DE4D   85 81       STA $81         take on as current sector
DE4F   60          RTS             Return from this subroutine
-----------------------------------------------------------------
[E4A6/DE57:CAA9,CF5A,E057,E065,E075/DE5E:C0BB,CAC3,DD81,DDF6,E047,E3B3]
[E3D4,E4ED/DE6C:DD67,E42D/DE73:DC92]
Give jobcodes to jobloop
DE50   A9 90       LDA #$90        Setjobcode for
DE52   8D 4D 02    STA $024D       'write sector'
DE55   D0 28       BNE $DE7F       Jump to $DE7F
DE57[5] A9 80      LDA #$80        Set 'read sector'
DE59   8D 4D 02    STA $024D       jobcode
DE5C   D0 21       BNE $DE7F       Jump to $DE7F
DE5E[8] A9 90      LDA #$90        Set 'write sector'
DE60   8D 4D 02    STA $024D       jobcode
DE63   D0 26       BNE $DE8B       Jump to $DE7F
DE65   A9 80       LDA #$80        Set jobcode to read
DE67   8D 4D 02    STA $024D       sector
DE6A   D0 1F       BNE $DE8B       Jump to $DE7F
DE6C[2] A9 90      LDA #$90        Set 'write sector'
DE6E   8D 4D 02    STA $024D       jobcode
DE71   D0 02       BNE $DE75       Jump to $DE7F
DE73[1] A9 80      LDA #$80        Set 'read sector'
```

```
DE75   8D 4D 02   STA $024D      jobcode
DE78   A6 82      LDX $82        Get current channel number
DE7A   B5 CD      LDA $CD,X      Find out number of third buffer
DE7C   AA         TAX            and save it down
DE7D   10 13      BPL $DE92      Is buffer used?
DE7F2  20 D0 D6   JSR $D6D0      NO—Track & sector to job loop
DE82   20 93 DF   JSR $DF93      Get number of current buffer and
DE85   AA         TAX            save it down
DE86   A5 7F      LDA $7F        Get number of current disk drive
DE88   9D 5B 02   STA $025B,X    and assign buffer
DE8B2  20 15 E1   JSR $E115      Clear 'buffer changed' flag
DE8E   20 93 DF   JSR $DF93      Get current buffer number & save
DE91   AA         TAX            it down
DE921  4C 06 D5   JMP $D506      Test parameters; execute job
--------------------------------------------------------------------
[DA34/E03F/E06B]
Parameters of next sector set by onhand linked bytes
DE95   A9 00      LDA #$00       Reset current buffer pointer to
DE97   20 C8 D4   JSR $D4C8      start-of-buffer
DE9A   20 37 D1   JSR $D137      Get byte from buffer and
DE9D   85 80      STA $80        take on as current track number
DE9F   20 37 D1   JSR $D137      Get byte from buffer and set as
DEA2   85 81      STA $81        current sector number
DEA4   60         RTS            Return from this subroutine
--------------------------------------------------------------------
[E467]
Copy file from buffer to another buffer
(Accumulator must contain # of bytes; Y the source buffers number;
X the destination buffer number)
DEA5   48         PHA            Save number of bytes to be copied
DEA6   A9 00      LDA #$00       Clear low-bytes of
DEA8   85 6F      STA $6F        both buffer
DEAA   85 71      STA $71        pointers
DEAC   B9 E0 FE   LDA $FEE0,Y    Set hi-byte of buffer address of
DEAF   85 70      STA $70        source buffer
DEB1   BD E0 FE   LDA $FEE0,X    Set hi-byte of buffer address of
DEB4   85 72      STA $72        destination buffer
DEB6   68         PLA            Get # of bytes to be transferred
DEB7   A8         TAY            again and save
DEB8   88         DEY            Initialize pointer
DEB91  B1 6F      LDA ($6F),Y    Read byte from source buffer and
DEBB   91 71      STA ($71),Y    transfer to destination buffer
DEBD   88         DEY            Set buffer pointer to next byte
DEBE   10 F9      BPL $DEB9      All data transferred?
DEC0   60         RTS            YES—Return from this subroutine
--------------------------------------------------------------------
```

```
[DD1A/E45B]
Clear buffer with $00 (Number in A)
DEC1  A8        TAY            Save buffer number
DEC2  B9 E0 FE  LDA $FEE0,Y    Get hi-byte of buffer address and
DEC5  85 70     STA $70        define in pointer
DEC7  A9 00     LDA #$00       Set low-byte of pointer to the
DEC9  85 6F     STA $6F        start-of-buffer
DECB  A8        TAY            Clear buffer with value of buffer
DECC1 91 6F     STA ($6F),Y    number
DECE  C8        INY            Buffer pointer to next position
DECF  D0 FB     BNE $DECC      Entire buffer cleared?
DED1  60        RTS            YES—Return from this subroutine
--------------------------------------------------------------------
[DF66/E1CB]
Get number of current side-sector
DED2  A9 00     LDA #$00       Get buffer address and set in
DED4  20 DC DE  JSR $DEDC      pointers $94/$95
DED7  A0 02     LDY #$02       Choose position in buffer
DED9  B1 94     LDA ($94),Y    Get # of side-sector from sector
DEDB  60        RTS            Return from this subroutine
--------------------------------------------------------------------
[DED4/DEEA/E41E/E46C]
Set buffer pointers $94/$95 to any position in buffer
DEDC  85 94     STA $94        Save desired position in buffer
DEDE  A6 82     LDX $82        Get current channel number and
DEE0  B5 CD     LDA $CD,X      establish preassigned 3rd buffer;
DEE2  AA        TAX            save buffer number
DEE3  BD E0 FE  LDA $FEE0,X    Get high-byte of buffer address
DEE6  85 95     STA $95        and set in pointer
DEE8  60        RTS            Return from this subroutine
--------------------------------------------------------------------
[DD33/DD57/DF14/E1FF/E35A/E3F4]
Set buffer pointer
DEE9  48        PHA            Save desired position in buffer
DEEA  20 DC DE  JSR $DEDC      Set buffer pointer
DEED  48        PHA            Save high-byte of buffer address
DEEE  8A        TXA            Get current buffr number & double
DEEF  0A        ASL A          it (table contains
DEF0  AA        TAX            2-byte values)
DEF1  68        PLA            Get hi-byte of buffer address and
DEF2  95 9A     STA $9A,X      set in buffer address table
DEF4  68        PLA            Get position in buffer & enter in
DEF5  95 99     STA $99,X      table
DEF7  60        RTS            Return from this subroutine
--------------------------------------------------------------------
```

```
[E258/E43C]
Read side-sector in buffer and set pointers
DEF8   20 66 DF    JSR $DF66        Test status of side-sectors
DEFB   30 0E       BMI $DF0B        Does a side-sector exist?
DEFD   50 13       BVC $DF12        YES-Is side-sector in buffer?
DEFF   A6 82       LDX $82          NO-Get current channel number
DF01   B5 CD       LDA $CD,X        Determine pre-arranged buffer #
DF03   20 1B DF    JSR $DF1B        Read side-sector into buffer
DF06   20 66 DF    JSR $DF66        Test status again
DF09   10 07       BPL $DF12        Everything runnng with no errors?
DF0B1  20 CB E1    JSR $E1CB        NO-Search for end of rel. file
DF0E   2C CE FE    BIT $FECE        Set N and V Processor flags
DF11   60          RTS              Return from this subroutine
DF122  A5 D6       LDA $D6          Get position in side-sector and
DF14   20 E9 DE    JSR $DEE9        set buffer pointer
DF17   2C CD FE    BIT $FECD        Clear all flags
DF1A   60          RTS              Return from this subroutine
-------------------------------------------------------------------
[DF03/E1EC/E4D6]
Read sector (buffer pointer to current buffr must turn according to
track & sector parameters of linked bytes)
DF1B   85 F9       STA $F9          Save buffer number
DF1D   A9 80       LDA #$80         Set 'read sector' jobode
DF1F   D0 04       BNE $DF25        Jump to $DF25
DF21   85 F9       STA $F9          Save current buffer number
DF23   A9 90       LDA #$90         Set 'write sector' jobcode and
DF251  48          PHA              note jobcode
DF26   B5 EC       LDA $EC,X        Get filetype of channel,determine
DF28   29 01       AND #$01         disk drive chosen
DF2A   85 7F       STA $7F          Take on as current drive number
DF2C   68          PLA              Get jobcode again and
DF2D   05 7F       ORA $7F          combine with drive number
DF2F   8D 4D 02    STA $024D        Save jobcode
DF32   B1 94       LDA ($94),Y      Read & store number of
DF34   85 80       STA $80          next sector from buffer
DF36   C8          INY              Set buffer pointer to next byte
DF37   B1 94       LDA ($94),Y      Get sector number from buffer and
DF39   85 81       STA $81          take it over
DF3B   A5 F9       LDA $F9          Current buffer number
DF3D   20 D3 D6    JSR $D6D3        Track/sector params to jobloop
DF40   A6 F9       LDX $F9          Get current buffer number
DF42   4C 93 D5    JMP $D593        Execute job
-------------------------------------------------------------------
```

```
[E3E9/E40F/E418]
Set side-sector pointer
DF45  A6 82      LDX $82         Current channel number
DF47  B5 CD      LDA $CD,X       Get number of preassigned buffer
DF49  4C EB D4   JMP $D4EB       Set buffer pointer
--------------------------------------------------------------------
[DF52/Jump to DF51/DF5C in DB48/DF4E/DF57/E381/DF51:DB48]
Calculate number of sectors in a relative file
DF4C  A9 78      LDA #$78        Add number of sector pointers per
DF4E  20 5C DF   JSR $DF5C       side- sector to pointer
DF51¹ CA         DEX             Set next side-sector number
DF52  10 F8      BPL $DF4C       All side-sectors considered?
DF54  A5 72      LDA $72         YES—Divide number of linked
DF56  4A         LSR A           bytes by 2
DF57  20 5C DF   JSR $DF5C       Add
DF5A  A5 73      LDA $73         number of side-sectors
DF5C² 18         CLC             Initialize addition
DF5D  65 70      ADC $70         Add value
DF5F  85 70      STA $70         to counter
DF61  90 02      BCC $DF65       Found a transfer?
DF63  E6 71      INC $71         YES—Correct high-byte
DF65¹ 60         RTS             Return from this subroutine
--------------------------------------------------------------------
[DEF8/DF06]
Test status of a side-sector
DF66  20 D2 DE   JSR $DED2       Get # of side-sectors from buffer
DF69  C5 D5      CMP $D5         Compare w/sector being searched
DF6B  D0 0E      BNE $DF7B       Is correct side-sector in buffer?
DF6D  A4 D6      LDY $D6         YES—Get pointer in buffer
DF6F  B1 94      LDA ($94),Y     Get track number of record
DF71  F0 04      BEQ $DF77       ist record applied?
DF73  2C CD FE   BIT $FECD       YES—Clear error flags
DF76  60         RTS             Return from this subroutine
DF77¹ 2C CF FE   BIT $FECF       Set 'no record' flag
DF7A  60         RTS             Return from this subroutine
DF7B¹ A5 D5      LDA $D5         Get # of side-sector searched
DF7D  C9 06      CMP #$06        Compare with largest side-sector
DF7F  B0 0A      BCS $DF8B       Is number in allowable range?
DF81  0A         ASL A           YES—Double side-sector number and
DF82  A8         TAY             save it
DF83  A9 04      LDA #$04        Set buffer number and
DF85  85 94      STA $94         store it
DF87  B1 94      LDA ($94),Y     Get track number of side-sector
```

```
DF89   D0 04      BNE $DF8F       Is track set?
DF8B1  2C D0 FE   BIT $FED0       NO—Set error flag
DF8E   60         RTS             Return from this subroutine
DF8F1  2C CE FE   BIT $FECE       Set 'Sector not in buffer' flag
DF92   60         RTS             Return from this subroutine
-----------------------------------------------------------------

[CAB7/CDEC/CF6F/CFF2/D0CC/D12F/D1D3/D324/D44D/D4CA/D4E8/D4F8/D6D0/D75E]
[DBC3/DDAB/DE2B/DE3E/DE82/DE8E/E07F/E457/E4D1/ECB3/ECDC/ED0D/ED32/ED46]
[EEF4]
Determine  number of current buffer
DF93   A6 82      LDX $82         Get number of current channel
DF95   B5 A7      LDA $A7,X       Test buffer layout
DF97   10 02      BPL $DF9B       Ist buffer reserved?
DF99   B5 AE      LDA $AE,X       Get number of second
DF9B   29 BF      AND #$BF        buffer
DF9D   60         RTS             Return from this subroutine
-----------------------------------------------------------------

[DDF1/E042/E10A/E115/E4B1]
Get current buffer status
DF9E   A6 82      LDX $82         Get number of present channel and
DFA0   8E 57 02   STX $0257       save it
DFA3   B5 A7      LDA $A7,X       Get buffer number
DFA5   10 09      BPL $DFB0       Is buffer reserved?
DFA7   8A         TXA             YES—Get channel number again
DFA8   18         CLC             and convert and save as
DFA9   69 07      ADC #$07        number for access to
DFAB   8D 57 02   STA $0257       2nd buffer
DFAE   B5 AE      LDA $AE,X       Get buffer status and test
DFB01  85 70      STA $70         Save status
DFB2   29 1F      AND #$1F        Mask out flags
DFB4   24 70      BIT $70         Is buffer active?
DFB6   60         RTS             Return from this subroutine
-----------------------------------------------------------------

[CF21/CF7E]
Test whether buffer is free
DFB7   A6 82      LDX $82         Current channel number
DFB9   B5 A7      LDA $A7,X       Get appropriate buffer number
DFBB   30 02      BMI $DFBF       Is buffer reserved?
DFBD   B5 AE      LDA $AE,X       YES—Test buffer status
DFBF1  C9 FF      CMP #$FF        Compare w/'buffer active' value
DFC1   60         RTS             Return from this subroutine
-----------------------------------------------------------------
```

```
[CF2E/CF88]
Activate current buffer (2-buffer operation)
DFC2   A6 82       LDX $82          Get current channel number
DFC4   09 80       ORA #$80         Set 'buffer inactive' flag
DFC6   B4 A7       LDY $A7,X        Get number of suitable buffer
DFC8   10 03       BPL $DFCD        Is 1st buffer reserved?
DFCA   95 A7       STA $A7,X        NO-Activate 1st buffer
DFCC   60          RTS              Return from this subroutine
DFCD1  95 AE       STA $AE,X        Assign # for 2nd buffr of channel
DFCF   60          RTS              Return from this subroutine
-----------------------------------------------------------------
[E153/E009:E291]
Write record of  a relative file
DFD0   A9 20       LDA #$20         Clear 'record full'
DFD2   20 9D DD    JSR $DD9D        flag
DFD5   A9 80       LDA #$80         Flag for last byte (EOI)
DFD7   20 A6 DD    JSR $DDA6        Test flag
DFDA   D0 41       BNE $701D        Last byte been received?
DFDC   A6 82       LDX $82          NO-Get current channel number and
DFDE   F6 B5       INC $B5,X        increment  recordd number
DFE0   D0 02       BNE $DFE4        Is a transfer imminent?
DFE2   F6 BB       INC $BB,X        YES-Correct high-byte
DFE41  A6 82       LDX $82          Get current channel number
DFE6   B5 C1       LDA $C1,X        Get pointer to position on buffer
DFE8   F0 2E       BEQ $7018        Pointer set?
DFEA   20 E8 D4    JSR $D4E8        YES-Get buffer pointer again
DFED   A6 82       LDX $82          Get current channel number and
DFEF   D5 C1       CMP $C1,X        compare buffr ptr w/record pointr
DFF1   90 03       BCC $DFF6        Is buffer pointer<record pointer?
DFF3   20 3C E0    JSR $E03C        NO-Write record to buffer
DFF61  A6 82       LDX $82          Current channel number
DFF8   B5 C1       LDA $C1,X        Get corresp. pointer to record &
DFFA   20 C8 D4    JSR $D4C8        set corresponding buffer pointer
DFFD   A1 99       LDA ($99,X)      Get filebyte from buffer
DFFF   85 85       STA $85          and save it
E001   A9 20       LDA #$20         Clear 'record full'
E003   20 9D DD    JSR $DD9D        flag
E006   20 04 E3    JSR $E304        Add record length to buffr pointr
E0091  48          PHA              Save new pointer value
E00A   90 28       BCC $E034        Record still pass in curnt sectr?
E00C   A9 00       LDA #$00         NO-Set position pointer and get
E00E   20 F6 D4    JSR $D4F6        byte (track number) from buffer
E011   D0 21       BNE $E034        Is there another fileblock ahead?
E013   68          PLA              NO-Get new buffer pointer,compare
E014   C9 02       CMP #$02         with value for file start
E016   F0 12       BEQ $E02A        Is the new buffer empty?
```

```
E018[1] A9 80      LDA #$80       NO—Set flag to last byte
E01A    20 97 DD   JSR $DD97      (EOI)
E01D[1] 20 2F D1   JSR $D12F      Determine buffer & channel number
E020    B5 99      LDA $99,X      Get lo-byte of buffer pointer &
E022    99 44 02   STA $0244,Y    save as last character
E025    A9 0D      LDA #$0D       Send <RETURN>
E027    85 85      STA $85        as output
E029    60         RTS            Return from this subroutine
E02A[1] 20 35 E0   JSR $E035      Set pointer to last character
E02D    A6 82      LDX $82        Number of current channel
E02F    A9 00      LDA #$00       Clear pointer to
E031    95 C1      STA $C1,X      next record
E033    60         RTS            Return from this subroutine
-----------------------------------------------------------------
[E00A/E011] Set pointer to last character
E034    68         PLA            Pointer to start of next record
E035[1] A6 82      LDX $82        Get current channel number & save
E037    95 C1      STA $C1,X      pointer
E039    4C 6E E1   JMP $E16E      Set pointer to last character
-----------------------------------------------------------------
[DFF3/E0A7/E135] Prepare sector of record
E03C    20 D3 D1   JSR $D1D3      Determine drive chosen
E03F    20 95 DE   JSR $DE95      Track/sector of next block
E042    20 9E DF   JSR $DF9E      Test buffer status
E045    50 16      BVC $E05D      Buffer contents been changed?
E047    20 5E DE   JSR $DE5E      YES—Write buffer to diskette
E04A    20 1E CF   JSR $CF1E      Adjust new buffer
E04D    A9 02      LDA #$02       Set buffer pointer to beginning
E04F    20 C8 D4   JSR $D4C8      OF file range
E052    20 AB DD   JSR $DDAB      Test last job for writing
E055    D0 24      BNE $E07B      Sector already been written on?
E057    20 57 DE   JSR $DE57      YES—Put sector back into buffer
E05A    4C 99 D5   JMP $D599      & wait til job has been executed
E05D[1] 20 1E CF   JSR $CF1E      Adjust new buffer
E060    20 AB DD   JSR $DDAB      Test last job for writing
E063    D0 06      BNE $E06B      Sector previously used f/writing?
E065    20 57 DE   JSR $DE57      YES—Read sector from disk & wait
E068    20 99 D5   JSR $D599      until job has  been executed
E06B[1] 20 95 DE   JSR $DE95      Track and sector of next block
E06E    A5 80      LDA $80        Get number of next track
E070    F0 09      BEQ $E07B      Another sector available?
E072    20 1E CF   JSR $CF1E      YES—Re-apply buffer
E075    20 57 DE   JSR $DE57      Read sector from diskette
E078    20 1E CF   JSR $CF1E      and apply new buffer
E07B[2] 60         RTS            Return from this subroutine
-----------------------------------------------------------------
```

```
[E0B4/E0FE]
Write a char. of a record into buffer
E07C  20 05 E1  JSR $E105     Set 'buffer altered' flag
E07F  20 93 DF  JSR $DF93     Get number of current buffer
E082  0A        ASL A         and double it (buffr pointr table
E083  AA        TAX           works with 2-byte values)
E084  A5 85     LDA $85       Get byte to be transferred
E086  81 99     STA ($99,X)   and write in buffer
E088  B4 99     LDY $99,X     Get buffer pointer (lo-byte)& set
E08A  C8        INY           to next position
E08B  D0 09     BNE $E096     Reached end of buffer?
E08D  A4 82     LDY $82       YES—Get present channel number &
E08F  B9 C1 00  LDA $00C1,Y   pointer to next record
E092  F0 0A     BEQ $E09E     Pointer set?
E094  A0 02     LDY #$02      YES—Set buffer pointer to start
E096¹ 98        TYA           of filerange
E097  A4 82     LDY $82       Get current channel number
E099  D9 C1 00  CMP $00C1,Y   Compare buffer- & record pointers
E09C  D0 05     BNE $E0A3     Record pointr at start-of-buffer?
E09E¹ A9 20     LDA #$20      YES—Set 'record full'
E0A0  4C 97 DD  JMP $DD97     flag
E0A3¹ F6 99     INC $99,X     Turn buffer pointer to next byte
E0A5  D0 03     BNE $E0AA     Reached end of buffer?
E0A7  20 3C E0  JSR $E03C     YES—Write sector to diskette
E0AA¹ 60        RTS           Return from this subroutine
------------------------------------------------------------------
[CFCB]
Write record to data buffer
E0AB  A9 A0     LDA #$A0      Test flags for 'last byte'(EOI) &
E0AD  20 A6 DD  JSR $DDA6     'record full'
E0B0  D0 27     BNE $E0D9     Is there a flag set?
E0B2¹ A5 85     LDA $85       NO—Get byte from input register &
E0B4  20 7C E0  JSR $E07C     write to record
E0B7  A5 F8     LDA $F8       Test for 'last byte' (EOI) flag
E0B9  F0 0D     BEQ $E0C8     Was that the last byte?
E0BB  60        RTS           YES—Return from this subroutine
E0BC¹ A9 20     LDA #$20      Test for 'record full'
E0BE  20 A6 DD  JSR $DDA6     flag
E0C1  F0 05     BEQ $E0C8     Is record already written full?
E0C3  A9 51     LDA #$51      YES—Set error flag
E0C5  8D 6C 02  STA $026C     for '51 overflow in record'
E0C8² 20 F3 E0  JSR $E0F3     Fill rest of record with nulls
E0CB  20 53 E1  JSR $E153     Get next record
E0CE  AD 6C 02  LDA $026C     Check for error flag
E0D1  F0 03     BEQ $E0D6     Encountered an error?
E0D3  4C C8 C1  JMP $C1C8     YES—Display error message
```

```
E0D6¹ 4C BC E6    JMP $E6BC        Prepare 'Ok' message
E0D9¹ 29 80       AND #$80         Test flag for 'last byte' (EOI)
E0DB  D0 05       BNE $E0E2        Is flag set?
E0DD  A5 F8       LDA $F8          NO—Test EOI from serial bus
E0DF  F0 DB       BEQ $E0BC        Is flag set?
E0E1  60          RTS              YES—Return from this subroutine
E0E2¹ A5 85       LDA $85          Get byte from input register;
E0E4  48          PHA              save it
E0E5  20 1C E3    JSR $E31C        Develop relative file
E0E8  68          PLA              Set back byte and
E0E9  85 85       STA $85          save it
E0EB  A9 80       LDA #$80         Clear 'last byte in file' (EOI)
E0ED  20 9D DD    JSR $DD9D        flag
E0F0  4C B2 E0    JMP $E0B2        Write record further in buffer
-------------------------------------------------------------------
[E0C8/E101]
Fill rest of record with empty bytes
E0F3  A9 20       LDA #$20         Test for 'record full'
E0F5  20 A6 DD    JSR $DDA6        flag
E0F8  D0 0A       BNE $E104        Is entire record filled?
E0FA  A9 00       LDA #$00         Set value for
E0FC  85 85       STA $85          null bytes
E0FE  20 7C E0    JSR $E07C        Write byte in record
E101  4C F3 E0    JMP $E0F3        Fill in next byte
E104¹ 60          RTS              Return from this subroutine
-------------------------------------------------------------------
[DE09/E07C]
Set flag for 'buffer data altered'
E105  A9 40       LDA #$40         Set flag for 'sector
E107  20 97 DD    JSR $DD97        altered'
E10A  20 9E DF    JSR $DF9E        Get buffer status
E10D  09 40       ORA #$40         Flag for 'buffer altered'
E10F  AE 57 02    LDX $0257        # of channel+7 (points to $AE)
E112  95 A7       STA $A7,X        Set buffer status anew
E114  60          RTS              Return from this subroutine
-------------------------------------------------------------------
[DE8B]
Clear 'buffer data altered' flag
E115  20 9E DF    JSR $DF9E        Get buffer status
E118  29 BF       AND #$BF         and combine with flag
E11A  AE 57 02    LDX $0257        Channel number for 2nd buffer
E11D  95 A7       STA $A7,X        Set buffer status again in table
E11F  60          RTS              Return from this subroutine
-------------------------------------------------------------------
```

```
[D3B1/E138:E294]
Get byte from record
E120  A9 80      LDA #$80       Check flag for
E122  20 A6 DD   JSR $DDA6      'last byte' (EOI)
E125  D0 37      BNE $E15E      Is it last byte of the record?
E127  20 2F D1   JSR $D12F      NO—Initialize buffer pointer
E12A  B5 99      LDA $99,X      Get buffer pointer and check with
E12C  D9 44 02   CMP $0244,Y    end position of record
E12F  F0 22      BEQ $E153      reached the end of record?
E131  F6 99      INC $99,X      NO—Buffer pointer to next byte
E133  D0 06      BNE $E13B      Is the data buffer full?
E135  20 3C E0   JSR $E03C      YES—Write sector; get next one
E138¹ 20 2F D1   JSR $D12F      Initialize buffer pointer
E13B¹ A1 99      LDA ($99,X)    Get byte from data buffer
E13D¹ 99 3E 02   STA $023E,Y    and save it
E140  A9 89      LDA #$89       Flag for read/write/EOI
E142  99 F2 00   STA $00F2,Y    set in channel status
E145  B5 99      LDA $99,X      Get lo-byte of buffer pointer and
E147  D9 44 02   CMP $0244,Y    compare w/value for end of record
E14A  F0 01      BEQ $E14D      Has entire record been read?
E14C  60         RTS            NO—Return from this subroutine
E14D¹ A9 81      LDA #$81       Set read/write flag in
E14F  99 F2 00   STA $00F2,Y    channel status
E152  60         RTS            Return from this subroutineE14A  F0 01
-------------------------------------------------------------------
[DCA3/E0CB/E12F]
Get record and output it
E153  20 D0 DF   JSR $DFD0      Get next record
E156  20 2F D1   JSR $D12F      Determine buffer- and channel #
E159  A5 85      LDA $85        Get byte and prepare
E15B  4C 3D E1   JMP $E13D      for output
-------------------------------------------------------------------
[E125/E262/E26F]
Error happens
E15E  A6 82      LDX $82        Get current channel # and
E160  A9 0D      LDA #$0D       conclude output
E162  9D 3E 02   STA $023E,X    with <RETURN>
E165  A9 81      LDA #$81       Set channel status
E167  95 F2      STA $F2,X      back again
E169  A9 50      LDA #$50       Display '50 Record
E16B  20 C8 C1   JSR $C1C8      Not Present' error message
-------------------------------------------------------------------
```

```
[E039]
Set pointer to last character of record
E16E   A6 82       LDX $82        Number of present channel
E170   B5 C1       LDA $C1,X      Pointer to start of next record
E172   85 87       STA $87        --get and save
E174   C6 87       DEC $87        Correct pointer(incl.0)& compare
E176   C9 02       CMP #$02       with value for start-of-file
E178   D0 04       BNE $E17E      Pointer at start of buffer?
E17A   A9 FF       LDA #$FF       YES—Set pointr to end of buffr &
E17C   85 87       STA $87        save it
E17E1  B5 C7       LDA $C7,X      Get record length and
E180   85 88       STA $88        save it
E182   20 E8 D4    JSR $D4E8      Set current buffer pointer
E185   A6 82       LDX $82        Get number of present channel
E187   C5 87       CMP $87        Compare buffer- w/record pointer
E189   90 19       BCC $E1A4      Is the buffer pointer larger?
E18B   F0 17       BEQ $E1A4      YES—Are both pointers equal?
E18D   20 1E CF    JSR $CF1E      NO—Apply new buffer
E190   20 B2 E1    JSR $E1B2      Look for end of record
E193   90 08       BCC $E19D      Find it?
E195   A6 82       LDX $82        NO—Get current channel number and
E197   9D 44 02    STA $0244,X    save pointer
E19A   4C 1E CF    JMP $CF1E      Apply new buffer and end
E19D1  20 1E CF    JSR $CF1E      Apply new buffer
E1A0   A9 FF       LDA #$FF       Set record pointer to end
E1A2   85 87       STA $87        of buffer
E1A42  20 B2 E1    JSR $E1B2      Search for end of record
E1A7   B0 03       BCS $E1AC      EEnd found?
E1A9   20 E8 D4    JSR $D4E8      YES—Set current buffer pointer
E1AC1  A6 82       LDX $82        Get number of matching channel &
E1AE   9D 44 02    STA $0244,X    save end position of records
E1B1   60          RTS            Return from this subroutine
-----------------------------------------------------------------
[E190/E1A4]
Search for end of record
E1B2   20 2B DE    JSR $DE2B      Set pointer to buffer start
E1B5   A4 87       LDY $87        Get current record pointer
E1B71  B1 94       LDA ($94),Y    Read byte from record
E1B9   D0 0D       BNE $E1C8      Byte = empty byte?
E1BB   88          DEY            YES—Move buffr pntr to next byte,
E1BC   C0 02       CPY #$02       compare with buffer begin. value
E1BE   90 04       BCC $E1C4      Reached start of buffer?
E1C0   C6 88       DEC $88        NO—Decrement record length
E1C2   D0 F3       BNE $E1B7      Entire record range searched?
```

```
E1C4¹ C6 88      DEC $88         YES—Correct record pointer
E1C6  18         CLC             Set 'end found' flag
E1C7  60         RTS             Return from this subroutine
E1C8¹ 98         TYA             Get current buffer position &
E1C9  38         SEC             set 'end not found' flag
E1CA  60         RTS             Return from this subroutine
-------------------------------------------------------------------
[CA56/CA69/DB32/DF0B/E31F]
Search for end of relative file
E1CB  20 D2 DE   JSR $DED2       Get number of current side-sector
E1CE  85 D5      STA $D5         and save it
E1D0  A9 04      LDA #$04        Reset buffer pointer to
E1D2  85 94      STA $94         beginning of sector
E1D4  A0 0A      LDY #$0A        Pointr to trak of last sidesector
E1D6  D0 04      BNE $E1DC       Jump to $E1DC
E1D8¹ 88         DEY             Set buffer pointr to track # of
E1D9  88         DEY             preceding side-sector
E1DA  30 26      BMI $E202       No more side-sectors on hand?
E1DC¹ B1 94      LDA ($94),Y     NO—Get track # of side-sector
E1DE  F0 F8      BEQ $E1D8       Is sector laid out?
E1E0  98         TYA             YES—Transmit side-sector
E1E1  4A         LSR A           number
E1E2  C5 D5      CMP $D5         and compare with current number
E1E4  F0 09      BEQ $E1EF       Identical?
E1E6  85 D5      STA $D5         NO—Save new side-sector number
E1E8  A6 82      LDX $82         Get number of present channel
E1EA  B5 CD      LDA $CD,X       Determine sector buffer and
E1EC  20 1B DF   JSR $DF1B       read sector
E1EF¹ A0 00      LDY #$00        Reset buffer pointer to begin. of
E1F1  84 94      STY $94         sector
E1F3  B1 94      LDA ($94),Y     Get track of next sector
E1F5  D0 0B      BNE $E202       No more side-sectors?
E1F7  C8         INY             YES—Set pointer to next position
E1F8  B1 94      LDA ($94),Y     Get # of applicable filebytes
E1FA  A8         TAY             and save them
E1FB  88         DEY             Set pointer to linked bytes of
E1FC  84 D6      STY $D6         last record's sector
E1FE  98         TYA             and save it
E1FF  4C E9 DE   JMP $DEE9       Set buffr pointr to this position
E202¹ A9 67      LDA #$67        Error message--
E204  20 45 E6   JSR $E645       '67 Illegal Track Or Sector'
-------------------------------------------------------------------
```

```
[Jump to routine at C146]
Record command routine ('P')
E207   20 B3 C2   JSR $C2B3       Set command string pointer
E20A   AD 01 02   LDA $0201       Get 2nd command char from buffer
E20D   85 83      STA $83         and set up as secondary address
E20F   20 EB D0   JSR $D0EB       Open read channel
E212   90 05      BCC $E219       Has a free channel been found?
E214   A9 70      LDA #$70        NO—Error message--
E216   20 C8 C1   JSR $C1C8       '70 No Channel'
E219[1] A9 A0     LDA #$A0        Clear EOI
E21B   20 9D DD   JSR $DD9D       flags
E21E   20 25 D1   JSR $D125       Get filetype and test it out
E221   F0 05      BEQ $E228       Is there a relative file?
E223   A9 64      LDA #$64        NO—Error message--
E225   20 C8 C1   JSR $C1C8       '64 File Type Mismatch'
E228[1] B5 EC     LDA $EC,X       Get channel flag and
E22A   29 01      AND #$01        take on chosen disk drive
E22C   85 7F      STA $7F         as current drive
E22E   AD 02 02   LDA $0202       Get 3rd char from input buffer &
E231   95 B5      STA $B5,X       set as low-byte of record number
E233   AD 03 02   LDA $0203       Get high-byte of record number &
E236   95 BB      STA $BB,X       take it up
E238   A6 82      LDX $82         Get number of present channel
E23A   A9 89      LDA #$89        Set read/write/EOI flag
E23C   95 F2      STA $F2,X       in channel status
E23E   AD 04 02   LDA $0204       Get 5th char from input buffer
E241   F0 10      BEQ $E253       No instructions?
E243   38         SEC             NO—Take up position in record
E244   E9 01      SBC #$01        and test for pointer=1
E246   F0 0B      BEQ $E253       Pointer set to start of record?
E248   D5 C7      CMP $C7,X       NO—Compare with record length
E24A   90 07      BCC $E253       Is position legal?
E24C   A9 51      LDA #$51        NO—Store '51 Overflow In Record'
E24E   8D 6C 02   STA $026C       in error flag
E251   A9 00      LDA #$00        Set position pointer to beginning
E253[3] 85 D4     STA $D4         of record
E255   20 0E CE   JSR $CE0E       Calculate position of record
E258   20 F8 DE   JSR $DEF8       Read in corresponding side-sector
E25B   50 08      BVC $E265       Side-sector read without errors?
E25D   A9 80      LDA #$80        NO—Set 'last byte' (EOI)
E25F   20 97 DD   JSR $DD97       flag
E262   4C 5E E1   JMP $E15E       Error—'50 Record Not Present'
```

```
E265¹ 20 75 E2   JSR $E275     Read record searched for
E268  A9 80      LDA #$80      Test for 'last byte'
E26A  20 A6 DD   JSR $DDA6     flag
E26D  F0 03      BEQ $E272     Record not onhand?
E26F  4C 5E E1   JMP $E15E     YES—Error—'50 Record Not Present'
E272¹ 4C 94 C1   JMP $C194     'Ok' message prepared
-------------------------------------------------------------------

[E265/E441]
Read record into buffer
E275  20 9C E2   JSR $E29C     Read sector containing record
E278  A5 D7      LDA $D7       Transfer position in record to
E27A  20 C8 D4   JSR $D4C8     current buffer pointer
E27D  A6 82      LDX $82       Present channel number
E27F  B5 C7      LDA $C7,X     Determne record length & subtract
E281  38         SEC           current position in data field
E282  E5 D4      SBC $D4       from record length
E284  B0 03      BCS $E289     Pointer still in field?
E286  4C 02 E2   JMP $E202     YES—'67 Illegal Track or Sector'
E289¹ 18         CLC           Figure position of desired bytes
E28A  65 D7      ADC $D7       in record
E28C  90 03      BCC $E291     Byte in next file sector?
E28E  69 01      ADC #$01      YES—Set to start position and set
E290  38         SEC           flag for next sector
E291¹ 20 09 E0   JSR $E009     Set pointer for next record
E294  4C 38 E1   JMP $E138     Get byte from record
E297  A9 51      LDA #$51      Error message--
E299  20 C8 C1   JSR $C1C8     '51 Overflow In Record'
-------------------------------------------------------------------

[CA6C/E322/E275]
Read record sector contained in buffer
E29C  A5 94      LDA $94       Retain current buffer pointer in
E29E  85 89      STA $89       temporary storage
E2A0  A5 95      LDA $95       in addresses
E2A2  85 8A      STA $8A       $89/$8A
E2A4  20 D0 E2   JSR $E2D0     Check buffer for sector
E2A7  D0 01      BNE $E2AA     Is the sector in buffer?
E2A9  60         RTS           YES—Return from this subroutine
E2AA¹ 20 F1 DD   JSR $DDF1     Write buffer contents to diskette
E2AD  20 0C DE   JSR $DE0C     Track /sector of next block
E2B0  A5 80      LDA $80       Get track number of next sector
E2B2  F0 0E      BEQ $E2C2     More sectors onhand?
E2B4  20 D3 E2   JSR $E2D3     YES—Test buffer for sector
E2B7  D0 06      BNE $E2BF     Is sector already in buffer?
E2B9  20 1E CF   JSR $CF1E     YES—Provide new buffer
E2BC  4C DA D2   JMP $D2DA     Free up all inactive buffers
E2BF1 20 DA D2   JSR $D2DA     Free up all inactive buffers
```

```
E2C2¹ A0 00      LDY #$00       Initialize buffer pointer
E2C4  B1 89      LDA ($89),Y    Get track number from side-sector
E2C6  85 80      STA $80        and take as current track no.
E2C8  C8         INY            Buffer pointer to next byte
E2C9  B1 89      LDA ($89),Y    Get number of file sector and
E2CB  85 81      STA $81        store it
E2CD  4C AF D0   JMP $D0AF      Read sector in buffer
-------------------------------------------------------------------
[E2A4]
Test to see whether sector is already in buffer
E2D0  20 3E DE   JSR $DE3E      Get track/sector of last job
E2D3¹ A0 00      LDY #$00       Initialize buffer pointer
E2D5  B1 89      LDA ($89),Y    Look for track from side-sector &
E2D7  C5 80      CMP $80        compare with last read value
E2D9  F0 01      BEQ $E2DC      Identical?
E2DB  60         RTS            NO—Return from this subroutine
E2DC¹ C8         INY            Set buffer pointer & sector #
E2DD  B1 89      LDA ($89),Y    Get # of sector being searched &
E2DF  C5 81      CMP $81        compare with current sector
E2E1  60         RTS            Return from this subroutine
-------------------------------------------------------------------
[DD7B/E3C2/E3CE]
Employ new record in sector
E2E2  20 2B DE   JSR $DE2B      Set current buffer address
E2E5  A0 02      LDY #$02       Pointer to begin. of file range
E2E7  A9 00      LDA #$00       Sector clear value
E2E9¹ 91 94      STA ($94),Y    Write empty byte to buffer
E2EB  C8         INY            Set buffer pointer to next byte
E2EC  D0 FB      BNE $E2E9      Entire buffer filled?
E2EE  20 04 E3   JSR $E304      YES—Get position of next record &
E2F1¹ 95 C1      STA $C1,X      save it
E2F3  A8         TAY            Take value as buffer pointer
E2F4  A9 FF      LDA #$FF       Value for opening record --
E2F6  91 94      STA ($94),Y    write to buffer
E2F8  20 04 E3   JSR $E304      Calculate position of next record
E2FB  90 F4      BCC $E2F1      Record still have room ?
E2FD  D0 04      BNE $E303      NO—Record passed in sector?
E2FF  A9 00      LDA #$00       YES—Setposition of next record to
E301  95 C1      STA $C1,X      start of next sector
E303¹ 60         RTS            Return from this subroutine
-------------------------------------------------------------------
```

```
[E006/E2EE/E2F8]
Calculate position of new record in sector
E304  A6 82      LDX $82        Get current channel number
E306  B5 C1      LDA $C1,X      and corresponding record pointer
E308  38         SEC            Set 'no more records' flag
E309  F0 0D      BEQ $E318      Fill in an old record?
E30B  18         CLC            NO—Add record length to
E30C  75 C7      ADC $C7,X      current position
E30E  90 0B      BCC $E31B      Record run to next sector?
E310  D0 06      BNE $E318      YES—Record fill entire sector?
E312  A9 02      LDA #$02       YES—Pointr to start of new sector
E314  2C CC FE   BIT $FECC      Set 'still another sector' flag
E317  60         RTS            Return from this subroutine
E318² 69 01      ADC #$01       Pointr to begin. of next record
E31A  38         SEC            Set 'no more sectors'  flag
E31B¹ 60         RTS            Return from this subroutine
-------------------------------------------------------------------
[E0E5/E33B:CA85]
Insert new records in relative file
E31C  20 D3 D1   JSR $D1D3      Get number of drive chosen
E31F  20 CB E1   JSR $E1CB      Get position of last record
E322  20 9C E2   JSR $E29C      Read side-sector and records
E325  20 7B CF   JSR $CF7B      Open new buffer
E328  A5 D6      LDA $D6        Retain pointer to file block
E32A  85 87      STA $87        in side-sector
E32C  A5 D5      LDA $D5        Temporarily store pointer of
E32E  85 86      STA $86        current side-sector
E330  A9 00      LDA #$00       Clear 'only one block'
E332  85 88      STA $88        flag
E334  A9 00      LDA #$00       Clear pointer of position
E336  85 D4      STA $D4        of record
E338  20 0E CE   JSR $CE0E      Calc. side-sector of fileblock
E33B¹ 20 4D EF   JSR $EF4D      Get number of blocks free
E33E  A4 82      LDY $82        Determine # off current channels
E340  B6 C7      LDX $C7,Y      Get corresponding record length &
E342  CA         DEX            correct it
E343  8A         TXA            (includes 0) and
E344  18         CLC            add to current
E345  65 D7      ADC $D7        buffer pointer
E347  90 0C      BCC $E355      Any new buffer pointer in sector?
E349  E6 D6      INC $D6        NO—Pointr in sidesector to track/
E34B  E6 D6      INC $D6        sector number of next fileblock
E34D  D0 06      BNE $E355      Pointr still in curr.sides-ector?
E34F  E6 D5      INC $D5        NO—Go to next side-sector
E351  A9 10      LDA #$10       Buffer pointr to begin. of track/
E353  85 D6      STA $D6        sector pointer of  fileblock
```

```
E355² A5 87      LDA $87      Old buffer pointer set to next
E357  18         CLC          sector file
E358  69 02      ADC #$02     (track/sector)
E35A  20 E9 DE   JSR $DEE9    Set buffer pointer
E35D  A5 D5      LDA $D5      Get # of current side-sectors and
E35F  C9 06      CMP #$06     compare with maximum value
E361  90 05      BCC $E368    Legal number?
E363² A9 52      LDA #$52     NO—Error --
E365  20 C8 C1   JSR $C1C8    '52 File Too Large'
E368¹ A5 D6      LDA $D6      Get current position in
E36A  38         SEC          side-sector
E36B  E5 87      SBC $87      Subtract last side-sector pointer
E36D  B0 03      BCS $E372    New value in preceding sector?
E36F  E9 0F      SBC #$0F     YES—Observe linking bytes at
E371  18         CLC          start of side-sector and
E372¹ 85 72      STA $72      save new value
E374  A5 D5      LDA $D5      Get current side-sector number &
E376  E5 86      SBC $86      and remove last number
E378  85 73      STA $73      Save new value
E37A  A2 00      LDX #$00     Clear temporary memory
E37C  86 70      STX $70      of number of blocks
E37E  86 71      STX $71      free
E380  AA         TAX          Side-sector number 0
E381  20 51 DF   JSR $DF51    Calculate # of blocks needed by
E384  A5 71      LDA $71      and hold it (high-byte)
E386  D0 07      BNE $E38F    Number of blocks < 256?
E388  A6 70      LDX $70      YES—Test low-byte
E38A  CA         DEX          of amount of blocks
E38B  D0 02      BNE $E38F    Just 1 block(side-sector)laidout?
E38D  E6 88      INC $88      Set 'just one block' flag
E38F² CD 73 02   CMP $0273    Compare w/ number of blocks free
E392  90 09      BCC $E39D    Any room left on the diskette?
E394  D0 CD      BNE $E363    NO—Any files past on disk?
E396  AD 72 02   LDA $0272    YES—Compare  lo-bytes of necess.
E399  C5 70      CMP $70      blocks with number of free blocks
E39B  90 C6      BCC $E363    File > capacity?
E39D¹ A9 01      LDA #$01     NO—Buffer pointer to sector #
E39F  20 F6 D4   JSR $D4F6    Get byte from buffer
E3A2  18         CLC          Increment pointer to current
E3A3  69 01      ADC #$01     filebyte in current sector
E3A5  A6 82      LDX $82      Get channel number
E3A7  95 C1      STA $C1,X    Save pointer to filebyte
E3A9  20 1E F1   JSR $F11E    Get next free sector from BAM
E3AC  20 FD DD   JSR $DDFD    Linking bytes for next sector
E3AF  A5 88      LDA $88      Flag for 'only one block'
E3B1  D0 15      BNE $E3C8    set?
```

```
E3B3    20 5E DE   JSR $DE5E    NO—Write sector to diskette
E3B6[2] 20 1E CF   JSR $CF1E    Changee buffer
E3B9    20 D0 D6   JSR $D6D0    Track/sector to job loop
E3BC    20 1E F1   JSR $F11E    Look for next free block in BAM
E3BF    20 FD DD   JSR $DDFD    Params of next block in buffer
E3C2    20 E2 E2   JSR $E2E2    Employ new record
E3C5    4C D4 E3   JMP $E3D4    Write sector to diskette
E3C8[2] 20 1E CF   JSR $CF1E    Change buffer
E3CB    20 D0 D6   JSR $D6D0    Track/sector to job loop
E3CE    20 E2 E2   JSR $E2E2    Use new record
E3D1    20 19 DE   JSR $DE19    Identify last sector
E3D4[1] 20 5E DE   JSR $DE5E    Write sector to diskette
E3D7    20 0C DE   JSR $DE0C    Track/sector from linking bytes
E3DA    A5 80      LDA $80      Get next track number and
E3DC    48         PHA          save it
E3DD    A5 81      LDA $81      Retain next
E3DF    48         PHA          sector number
E3E0    20 3E DE   JSR $DE3E    Get track/sector of last job
E3E3    A5 81      LDA $81      Save last sector
E3E5    48         PHA          number
E3E6    A5 80      LDA $80      Retain number of last
E3E8    48         PHA          sector
E3E9    20 45 DF   JSR $DF45    Set buffr pointr f/side-sector &
E3EC    AA         TAX          save low-byte
E3ED    D0 0A      BNE $E3F9    Pointer at buffer start?
E3EF    20 4E E4   JSR $E44E    YES—Open new side-sector
E3F2    A9 10      LDA #$10     Buffr pointr to begin. of pointer
E3F4    20 E9 DE   JSR $DEE9    of file sectors
E3F7    E6 86      INC $86      Increment side-sector number
E3F9[1] 68         PLA          Get track of last sector and
E3FA    20 8D DD   JSR $DD8D    enter  in side-sector
E3FD    68         PLA          Get sector number and take
E3FE    20 8D DD   JSR $DD8D    byte in side-sector
E401    68         PLA          Get current sector number
E402    85 81      STA $81      and store it
E404    68         PLA          Get current track number and
E405    85 80      STA $80      store it
E407    F0 0F      BEQ $E418    Last block?
E409    A5 86      LDA $86      NO—Compare current side-sector #
E40B    C5 D5      CMP $D5      with the last one
E40D    D0 A7      BNE $E3B6    Changed?
E40F    20 45 DF   JSR $DF45    YES—Position buffer pointer and
E412    C5 D6      CMP $D6      compare with side-sector pointer
E414    90 A0      BCC $E3B6    Is buffer pointer less?
E416    F0 B0      BEQ $E3C8    NO—Is it equal?
```

```
E418¹ 20 45 DF   JSR $DF45      NO—Position buffer pointer and
E41B  48         PHA            save as ending
E41C  A9 00      LDA #$00       Reset buffer pointer
E41E  20 DC DE   JSR $DEDC      to zero
E421  A9 00      LDA #$00       Set buffer pointer  to
E423  A8         TAY            beginning of sector
E424  91 94      STA ($94),Y    Set flag to last block in buffer
E426  C8         INY            Set buffer pointer to next byte
E427  68         PLA            Set pointer to end and
E428  38         SEC            decrement
E429  E9 01      SBC #$01       by one
E42B  91 94      STA ($94),Y    Pointer—number of good bytes
E42D  20 6C DE   JSR $DE6C      Write sector to diskette
E430  20 99 D5   JSR $D599      and test for write error
E433  20 F4 EE   JSR $EEF4      Put sector in BAM
E436  20 0E CE   JSR $CE0E      Re-initialize REL file pointer
E439  20 1E CF   JSR $CF1E      Get another buffer
E43C  20 F8 DE   JSR $DEF8      Check side-sector
E43F  70 03      BVS $E444      Is correct side-sector in buffer?
E441  4C 75 E2   JMP $E275      YES—Record pointer reset; end
E444¹ A9 80      LDA #$80       Reset filetype
E446  20 97 DD   JSR $DD97      pointer and flags
E449  A9 50      LDA #$50       Display error message --
E44B  20 C8 C1   JSR $C1C8      '50 Record Not Present'
-----------------------------------------------------------------
[E3EF]
Prepare new side-sector
E44E  20 1E F1   JSR $F11E      Determine next free block
E451  20 1E CF   JSR $CF1E      Choose buffer
E454  20 F1 DD   JSR $DDF1      Write previous side-sector
E457  20 93 DF   JSR $DF93      Get buffer number
E45A  48         PHA            and save it
E45B  20 C1 DE   JSR $DEC1      Clear file buffer
E45E  A6 82      LDX $82        Channel number
E460  B5 CD      LDA $CD,X      Take channel number for
E462  A8         TAY            side-sector
E463  68         PLA            from stack and save in
E464  AA         TAX            X/Y-registers
E465  A9 10      LDA #$10       Take 16 byts of previos sidesectr
E467  20 A5 DE   JSR $DEA5      into current buffer
E46A  A9 00      LDA #$00       Buffer pointer value
E46C  20 DC DE   JSR $DEDC      Reset buffer pointer
E46F  A0 02      LDY #$02       Take buffr of previos side-sector
E471  B1 94      LDA ($94),Y    and get side-sector number
E473  48         PHA            Save number of last side-sector
E474  A9 00      LDA #$00       Turn buffr pointer to buffr for
E476  20 C8 D4   JSR $D4C8      new side-sector and set back
```

```
E479  68        PLA         Get number of last side-sector
E47A  18        CLC         Increase by one
E47B  69 01     ADC #$01    and store
E47D  91 94     STA ($94),Y as new number
E47F  0A        ASL A       Double value
E480  69 04     ADC #$04    and add 4
E482  85 89     STA $89     Set track/sector pointer
E484  A8        TAY         and save it;
E485  38        SEC         from that, compute
E486  E9 02     SBC #$02    the pointer to
E488  85 8A     STA $8A     the previous side-sector
E48A  A5 80     LDA $80     Save
E48C  85 87     STA $87     track number
E48E  91 94     STA ($94),Y Write to current buffer
E490  C8        INY         Set buffer pointer to next byte
E491  A5 81     LDA $81     Store sector number
E493  85 88     STA $88     and take into
E495  91 94     STA ($94),Y current buffer
E497  A0 00     LDY #$00    Set buffer pointer to
E499  98        TYA         beginning of sector
E49A  91 94     STA ($94),Y Flag for last side-sector
E49C  C8        INY         Set buffer pointer to next byte
E49D  A9 11     LDA #$11    Set number of good bytes to
E49F  91 94     STA ($94),Y sector (17)
E4A1  A9 10     LDA #$10    Put buffer pointer
E4A3  20 C8 D4  JSR $D4C8   to position 16
E4A6  20 50 DE  JSR $DE50   Write sector to diskette
E4A9  20 99 D5  JSR $D599   Wait f/messge frm diskcontroller
E4AC  A6 82     LDX $82     Set current channel number
E4AE  B5 CD     LDA $CD,X   Get # of buffer to side-sector
E4B0  48        PHA         and save it down
E4B1  20 9E DF  JSR $DF9E   Get buffer number
E4B4  A6 82     LDX $82     Current channel number;
E4B6  95 CD     STA $CD,X   save as third buffer
E4B8  68        PLA         Buffer number for side-sector
E4B9  AE 57 02  LDX $0257   Pntr to last active file buffer
E4BC  95 A7     STA $A7,X   Lay out buffer
E4BE  A9 00     LDA #$00    Reset buffer pointer
E4C0  20 C8 D4  JSR $D4C8   to zero
E4C3  A0 00     LDY #$00    Buffer pointer to start-of-sector
E4C5  A5 80     LDA $80     Take track number into
E4C7  91 94     STA ($94),Y file buffer
E4C9  C8        INY         Set pointer to next character
E4CA  A5 81     LDA $81     Take buffer number
E4CC  91 94     STA ($94),Y into the buffer
E4CE  4C DE E4  JMP $E4DE   Write side-sector to diskette
E4D1[1] 20 93 DF  JSR $DF93   Determine current buffer number
```

```
E4D4   A6 82      LDX $82         Current channel number
E4D6   20 1B DF   JSR $DF1B       Read next side-sector from disk
E4D9   A9 00      LDA #$00        Reset buffer pointer
E4DB   20 C8 D4   JSR $D4C8       to zero
E4DE1  C6 8A      DEC $8A         Correct side-
E4E0   C6 8A      DEC $8A         sector number
E4E2   A4 89      LDY $89         Buffr pntr for track/sector pos.
E4E4   A5 87      LDA $87         Write track number
E4E6   91 94      STA ($94),Y     to the file buffer
E4E8   C8         INY             Set buffer pointer to next byte
E4E9   A5 88      LDA $88         Get sector number and take
E4EB   91 94      STA ($94),Y     into the buffer
E4ED   20 5E DE   JSR $DE5E       Write side-sector to diskette
E4F0   20 99 D5   JSR $D599       Wait f/messge frm diskcontroller
E4F3   A4 8A      LDY $8A         Get side-sector number and
E4F5   C0 03      CPY #$03        test it
E4F7   B0 D8      BCS $E4D1       Greater than 3?
E4F9   4C 1E CF   JMP $CF1E       NO—Choose another buffer
-----------------------------------------------------------------
The first byte is the error number in BCD-Code. Next follows the
text of the error msg. The start and ending of these text strings
are indicated by bit7 in the first & last byte set to 1. Some
values are set up as short codes. The most significant byte-half of
these values is 0.  They are handled like error messages.
-----------------------------------------------------------------
E4FC 00                            'ok'
E4FD A0 4F CB
-----------------------------------------------------------------
E500 20 21 22 23 24 27             'read error'
E506 D2 45 41 44 89
-----------------------------------------------------------------
E50B 52                            'file too large'
E50C 83 20 54 4F 4F 20 4C 41 52 47 C5
-----------------------------------------------------------------
E517 50                            'record not present'
E518 8B 06 20 50 52 45 53 45 4E D4
-----------------------------------------------------------------
E522 51                            'overflow in record'
E523 CF 56 45 52 46 4C 4F 57 20 49 4E 8B
-----------------------------------------------------------------
E52F 25 28                         'write error'
E531 8A 89
-----------------------------------------------------------------
E533 26                            'write protect on'
E534 8A 20 50 52 4F 54 45 43 54 20 4F CE
-----------------------------------------------------------------
```

```
E540 29                          'disk id mismatch'
E541 88 20 49 44 85
-------------------------------------------------------------------
E546 30 31 32 33 34              'syntax error'
E54B D3 59 4E 54 41 58 89
-------------------------------------------------------------------
E552 60                          'write file open'
E553 8A 03 84
-------------------------------------------------------------------
E556 63                          'file exists'
E557 83 20 45 58 49 53 54 D3
-------------------------------------------------------------------
E55F 64                          'file type mismatch'
E560 83 20 54 59 50 45 85
-------------------------------------------------------------------
E567 65                          'no block'
E568 CE 4F 20 42 4C 4F 43 CB
-------------------------------------------------------------------
E570 66 67                       'illegal track or sector'
E572 C9 4C 4C 45 47 41 4C 20 54 52 41 43 4B 20 4F 52 20 53 45 43 54 4F D2
-------------------------------------------------------------------
E589 61                          'file not open'
E58A 83 06 84
-------------------------------------------------------------------
E58D 39 62                       'file not found'
E58F 83 06 87
-------------------------------------------------------------------
E592 01                          'files scratched'
E593 83 53 20 53 43 52 41 54 43 48 45 C4
-------------------------------------------------------------------
E59F 70                          'no channel'
E5A0 CE 4F 20 43 48 41 4E 4E 45 CC
-------------------------------------------------------------------
E5AA 71                          'dir error'
E5AB C4 49 52 89
-------------------------------------------------------------------
E5AF 72                          'disk full'
E5B0 88 20 46 55 4C CC
-------------------------------------------------------------------
E5B6 73                          'cbm dos v3.0 1571'
E5B7 C3 42 4D 20 44 4F 53 20 56 33 2E 30 20 31 35 37 B1
-------------------------------------------------------------------
E5C8 74                          'drive not ready'
E5C9 C4 52 49 56 45 06 20 52 45 41 44 D9
-------------------------------------------------------------------
```

```
Often-used words and their short codes :
--------------------------------------------------------------------
E5D5 09                                'error'
E5D6 C5 52 52 4F D2
--------------------------------------------------------------------
E5DB 0A                                'write'
E5DC D7 52 49 54 C5
--------------------------------------------------------------------
E5E1 03                                'file'
E5E2 C6 49 4C C5
--------------------------------------------------------------------
E5E6 04                                'open'
E5E7 CF 50 45 CE
--------------------------------------------------------------------
E5EB 05                                'mismatch'
E5EC CD 49 53 4D 41 54 43 C8
--------------------------------------------------------------------
E5F4 06                                'not'
E5F5 CE 4F D4
--------------------------------------------------------------------
E5F8 07                                'found'
E5F9 C6 4F 55 4E C4
--------------------------------------------------------------------
E5FE 08                                'disk'
E5FF C4 49 53 CB
--------------------------------------------------------------------
E603 0B                                'record'
E604 D2 45 43 4F 52 C4
--------------------------------------------------------------------
[8391/91AA/A47B/A6CB/BF63/C8EC/D641/E60D:A9D2]
Error message output
(A must contain error no.; X the buffer number)
E60A  4C B9 A9   JMP $A9B9       1571 mode observed
E60D  8A         TXA             Double buffer
E60E  0A         ASL A           number and set
E60F  AA         TAX             as pointer on disk controller
E610  B5 06      LDA $06,X       Get track # from disk controller
E612  85 80      STA $80         and store it
E614  B5 07      LDA $07,X       Get sector number and
E616  85 81      STA $81         store it
E618  68         PLA             Get error number ready again
E619  29 0F      AND #$0F        Is the error number
E61B  F0 08      BEQ $E625       15 or greater?
E61D  C9 0F      CMP #$0F        YES—Is it equal to
E61F  D0 06      BNE $E627       error number 15?
E621  A9 74      LDA #$74        YES—Internal # of error message
E623  D0 08      BNE $E62D       Jump to $E62D
```

```
E625¹ A9 06     LDA #$06     Convert error number
E627¹ 09 20     ORA #$20     for read error and correct
E629  AA        TAX          for error
E62A  CA        DEX          table
E62B  CA        DEX          (BCD codes)
E62C  8A        TXA          Repeat number
E62D¹ 48        PHA          Retain error number
E62E  AD 2A 02  LDA $022A    Number of command begin executed
E631  C9 00     CMP #$00     Compare with 'validate' command
E633  D0 0F     BNE $E644    Identical?
E635  A9 FF     LDA #$FF     YES—Then clear
E637  8D 2A 02  STA $022A    command number and
E63A  68        PLA          return an error number
E63B  20 C7 E6  JSR $E6C7    Put error message in
E63E  20 42 D0  JSR $D042    Initialize - command execute
E641  4C 48 E6  JMP $E648    Activate error messages
E644¹ 68        PLA          Get back error number
-----------------------------------------------------------------
[A582/A9F5/CD2E/D54F/D577/DC03/E204/E829/F1DC/F1F7/F248]
Prepare error message
E645  20 C7 E6  JSR $E6C7    Produce error message in buffer
-----------------------------------------------------------------
[A4AA/D021/E641/F01F]
Activate error message
E648  20 BD C1  JSR $C1BD    Clr input buffer f/command string
E64B  A9 00     LDA #$00     Write back to BAM by hindering
E64D  8D F9 02  STA $02F9    flag setting
E650  20 2C C1  JSR $C12C    LED blinks
E653  20 DA D4  JSR $D4DA    Close channel
E656  A9 00     LDA #$00     Reset pointer to position
E658  85 A3     STA $A3      in command string
E65A  A2 45     LDX #$45     Reset
E65C  9A        TXS          stack pointer
E65D  A5 84     LDA $84      Find out standard
E65F  29 0F     AND #$0F     secondary address and
E661  85 83     STA $83      save it down
E663  C9 0F     CMP #$0F     Compare with channel 15
E665  F0 31     BEQ $E698    Is it the command channel?
E667  78        SEI          NO—Disable disk controller
E668  A5 79     LDA $79      'Listen found'
E66A  D0 1C     BNE $E688    flag active?
E66C  A5 7A     LDA $7A      NO—What above the 'Talk found'
E66E  D0 10     BNE $E680    flag?
E670  A6 83     LDX $83      NO—Get secondary address and
E672  BD 2B 02  LDA $022B,X  test appropriate
E675  C9 FF     CMP #$FF     channel status
E677  F0 1F     BEQ $E698    Is the channel active?
```

```
E679   29 0F      AND #$0F         YES—Prep channel number and
E67B   85 82      STA $82          store it
E67D   4C 8E E6   JMP $E68E        for a wait loop
E680¹  20 EB D0   JSR $D0EB        Get channel number
E683   EA         NOP              Empty space..............
E684   EA         NOP              [Resulting from modification]
E685   EA         NOP              [of 1541 ROM]
E686   D0 06      BNE $E68E        Jump to $E68E
E688¹  20 07 D1   JSR $D107        Get write channel
E68B   EA         NOP              Empty space.............
E68C   EA         NOP              [Due to modification]
E68D   EA         NOP              [of 1541 ROM]
E68E²  20 25 D1   JSR $D125        Determine current filetype
E691   C9 04      CMP #$04         Test for relative file
E693   B0 03      BCS $E698        Is it s relative file?
E695   20 27 D2   JSR $D227        NO—Free up all channels for
E698⁵  4C 6B 83   JMP $836B        command wait loop
----------------------------------------------------------------
[E6EA/E6F4]
Convert a binary number to a BCD number
E69B   AA         TAX              Save binary number
E69C   A9 00      LDA #$00         Set accumulator back
E69E   F8         SED              [Error -- see Chapter 7.1.5]
E69F¹  E0 00      CPX #$00         Compare binary value and 0
E6A1   F0 07      BEQ $E6AA        Identical?
E6A3   18         CLC              Get addition ready
E6A4   69 01      ADC #$01         Add X times 1 in
E6A6   CA         DEX              BCD mode
E6A7   4C 9F E6   JMP $E69F        Count up until X=0
E6AA¹  D8         CLD              Turn off decimal mode
----------------------------------------------------------------
[E6D1]
Convert BCD number into two ASCII-characters
E6AB   AA         TAX              Save BCD value
E6AC   4A         LSR A            Isolate most significant
E6AD   4A         LSR A            nibble; first digit
E6AE   4A         LSR A            prepares
E6AF   4A         LSR A            BCD number
E6B0   20 B4 E6   JSR $E6B4        --convert to  ASCII value
E6B3   8A         TXA              Get original value again and
E6B4¹  29 0F      AND #$0F         isolate 2nd BCD number
E6B6   09 30      ORA #$30         Convert to ASCII and write
E6B8   91 A5      STA ($A5),Y      in current buffer
E6BA   C8         INY              Pointer to next byte in buffer
E6BB   60         RTS              Return from this subroutine
----------------------------------------------------------------
```

```
[C150/E0D6]
Prepare '00 OK' error message
E6BC  20 23 C1   JSR $C123       Reset error flags
E6BF  A9 00      LDA #$00        Error number  for'OK'
-----------------------------------------------------------------
[D24A/EBD7]
Output error message with track & sector =0
E6C1  A0 00      LDY #$00        Track number and
E6C3  84 80      STY $80         Sector number
E6C5  84 81      STY $81         cleared
-----------------------------------------------------------------
[C1A7/E63B/E645/EFCB]
Produce error message in buffer (number in accumulator)
E6C7  A0 00      LDY #$00        Set pointer to position in buffer
E6C9  A2 D5      LDX #$D5        Save buffer address of
E6CB  86 A5      STX $A5         error message buffer ($02D5) in
E6CD  A2 02      LDX #$02        pointers
E6CF  86 A6      STX $A6         $A5/$A6
E6D1  20 AB E6   JSR $E6AB       Write error number in buffer
E6D4  A9 2C      LDA #$2C        Take up comma (,) after
E6D6  91 A5      STA ($A5),Y     error number in buffer
E6D8  C8         INY             Set buffer pointer to next byte
E6D9  AD D5 02   LDA $02D5       Copy first digit of error number
E6DC  8D 43 02   STA $0243       into output register
E6DF  8A         TXA             Repeat error number
E6E0  20 06 E7   JSR $E706       Write error in text form
E6E3  A9 2C      LDA #$2C        to buffer, and set in
E6E5  91 A5      STA ($A5),Y     trailing comma
E6E7  C8         INY             Set buffer pointer to next byte
E6E8  A5 80      LDA $80         Convert track number where error
E6EA  20 9B E6   JSR $E69B       occurd into ASCII;put into buffer
E6ED  A9 2C      LDA #$2C        Set comma(,) into buffer
E6EF  91 A5      STA ($A5),Y     as separating character
E6F1  C8         INY             Buffer pointer to next byte
E6F2  A5 81      LDA $81         Convert sector number where error
E6F4  20 9B E6   JSR $E69B       occurd into ASCII;put into buffer
E6F7  88         DEY             Calculate length
E6F8  98         TYA             of error message
E6F9  18         CLC             in buffer and
E6FA  69 D5      ADC #$D5        save
E6FC  8D 49 02   STA $0249       it down
E6FF  E6 A5      INC $A5         Buffer ptr. ($A5/$A6) to 2nd char
E701  A9 88      LDA #$88        Set 'ready for output'
E703  85 F7      STA $F7         flag and
E705  60         RTS             return from this subroutine
-----------------------------------------------------------------
```

```
[E6E0/E75F]
Write error message in text form to error buffer
E706   AA           TAX              Save error number
E707   A5 86        LDA $86          The value which will be used in
E709   48           PHA              temporary storage will be
E70A   A5 87        LDA $87          retained, since this
E70C   48           PHA              address is needed for the routine
E70D   A9 FC        LDA #$FC         Sett address for
E70F   85 86        STA $86          beginning of text
E711   A9 E4        LDA #$E4         table ($E4FC) into
E713   85 87        STA $87          pointers $86/$87
E715   8A           TXA              Get error number again
E716   A2 00        LDX #$00         Initialize buffer pointer
E718¹  C1 86        CMP ($86,X)      Compare number with text table
E71A   F0 21        BEQ $E73D        Identical?
E71C   48           PHA              NO—Save error number
E71D   20 75 E7     JSR $E775        Increment buffer pointer
E720   90 05        BCC $E727        Jump to $E727
E722¹  20 75 E7     JSR $E775        Increment buffer pointer
E725   90 FB        BCC $E722        Jump to $E727
E727¹  A5 87        LDA $87          Get high-byte of text pointer;
E729   C9 E6        CMP #$E6         test for end value
E72B   90 08        BCC $E735        Reached the end of the table?
E72D   D0 0A        BNE $E739        YES—Same memory page reached?
E72F   A9 0A        LDA #$0A         YES—Compare lo-byte of textpointr
E731   C5 86        CMP $86          with end value
E733   90 04        BCC $E739        Reached end of error table?
E735¹  68           PLA              YES—Repeat error number
E736   4C 18 E7     JMP $E718        Search for error number
E739²  68           PLA              Get error numbe again
E73A   4C 4D E7     JMP $E74D        End
E73D²  20 67 E7     JSR $E767        Get byte from error text
E740   90 FB        BCC $E73D        Start-flag set?
E742¹  20 54 E7     JSR $E754        YES—Write char. to buffer
E745   20 67 E7     JSR $E767        Get byte from error text
E748   90 F8        BCC $E742        Is end flag set?
E74A   20 54 E7     JSR $E754        YES—Write character into buffer
E74D¹  68           PLA              Get zeropage value again and
E74E   85 87        STA $87          get original
E750   68           PLA              value
E751   85 86        STA $86          ready again
E753   60           RTS              Return from this subroutine
---------------------------------------------------------------
```

```
[E742/E74A]
Write ASCII character into buffer
Non-ASCII characters will be interpreted as error numbers
E754   C9 20      CMP #$20       Compare with space
E756   B0 0B      BCS $E763      Is character > space?
E758   AA         TAX            NO—Save char. as error number and
E759   A9 20      LDA #$20       write space into
E75B   91 A5      STA ($A5),Y    current buffer position
E75D   C8         INY            Set buffer pointer to next char &
E75E   8A         TXA            get error number again
E75F   20 06 E7   JSR $E706      Error message text to buffer
E762   60         RTS            Return from this subroutine
E763¹  91 A5      STA ($A5),Y    Write ASCII char to buffer and
E765   C8         INY            set buffr pointr to next position
E766   60         RTS            Return from this subroutine
--------------------------------------------------------------------
[E73D/E745]
Get a character of error text from the text table
E767   E6 86      INC $86        Text pointer to next character
E769   D0 02      BNE $E76D      Has a transfer occurred ?
E76B   E6 87      INC $87        YES—Correct high-byte
E76D²  A1 86      LDA ($86,X)    Get character from text table
E76F   0A         ASL A          Bit7 in carry
E770   A1 86      LDA ($86,X)    Get original char one more time
E772   29 7F      AND #$7F       Bit7 masked
E774   60         RTS            Return from this subroutine
--------------------------------------------------------------------
[E71D/E722]
Get current byte from table
E775   20 6D E7   JSR $E76D      Get character from table
E778   E6 86      INC $86        Text pointer turns to next byte
E77A   D0 02      BNE $E77E      Has there been a transfer?
E77C   E6 87      INC $87        YES—Correct high-byte
E77E¹  60         RTS            Return from this subroutine
--------------------------------------------------------------------
```

```
[Not used in 1571 DOS]
E77F   60         RTS          Earlier 1541 ROM vrsions had
E780   60         RTS          an Autoboot routine here
E781   EA ...     NOP          Eventual jump in ths routine
E7A1   ... EA     NOP          is caught and ended here by
E7A2   60         RTS          the 1571 drive
-----------------------------------------------------------------
[Jump through routine C146]
Routine for &-Command [AUTOSTART program]
E7A3   20 FE A5   JSR $A5FE    Patch (=CORRECTION) for 1571 DOS
E7A6   EA         NOP          [No operation, modified for
E7A7   EA         NOP          1541 ROM]
E7A8   20 58 F2   JSR $F258    No function (RTS)
E7AB   AD 78 02   LDA $0278    No. of filenames marked for
E7AE   48         PHA          Data entry
E7AF   A9 01      LDA #$01     Limit work to
E7B1   8D 78 02   STA $0278    First file
E7B4   A9 FF      LDA #$FF     Entry flag
E7B6   85 86      STA $86      Cleared
E7B8   20 4F C4   JSR $C44F    Look in directory/file entry
E7BB   AD 80 02   LDA $0280    Flag/search result (track #)
E7BE   D0 05      BNE $E7C5    File entry found?
E7C0   A9 39      LDA #$39     Error message
E7C2   20 C8 C1   JSR $C1C8    "39 file not found" displayd
E7C5[1] 68        PLA          Number of filenames repeated
E7C6   8D 78 02   STA $0278    and reset
E7C9   AD 80 02   LDA $0280    Track of first file sector
E7CC   85 80      STA $80      transferred
E7CE   AD 85 02   LDA $0285    First sector number
E7D1   85 81      STA $81      transferred
E7D3   A9 03      LDA #$03     Identifier for USR data
E7D5   20 77 D4   JSR $D477    opened; first sector read in
E7D8[1] A9 00     LDA #$00     Check sum
E7DA   85 87      STA $87      Clear
E7DC   20 39 E8   JSR $E839    Get starting memory address
E7DF   85 88      STA $88      from buffer
E7E1   20 4B E8   JSR $E84B    and put into point $88/89;
E7E4   20 39 E8   JSR $E839    Compare this
E7E7   85 89      STA $89      value with
E7E9   20 4B E8   JSR $E84B    Checksum
E7EC   A5 86      LDA $86      Flag/"Starting address read"
E7EE   F0 0A      BEQ $E7FA    set ?
E7F0   A5 88      LDA $88      No, note starting address
E7F2   48         PHA          Low byte and
E7F3   A5 89      LDA $89      Note starting address
E7F5   48         PHA          Low byte
E7F6   A9 00      LDA #$00     Set flag for "Starting
```

```
E7F8  85 86       STA $86         address read"
E7FA[1] 20 39 E8  JSR $E839       No. of data bytes following
E7FD  85 8A       STA $8A         Gotten from buffer and noted
E7FF  20 4B E8    JSR $E84B       Initialize checksum
E802[1] 20 39 E8  JSR $E839       Get data byte from Buffer
E805  A0 00       LDY #$00        Initialze memory pointer
E807  91 88       STA($88),Y      and store byte
E809  20 4B E8    JSR $E84B       Byte taken in checksum
E80C  A5 88       LDA $88         Memory address (low byte)
E80E  18          CLC             The address at which data is
E80F  69 01       ADC #$01        stored should be incremented
E811  85 88       STA $88         by one
E813  90 02       BCC $E817       Anything being transferred?
E815  E6 89       INC $89         Yes, correct high byte
E817[1] C6 8A     DEC $8A         Counter/total # data bytes
E819  D0 E7       BNE $E802       All bytes in memory?
E81B  20 35 CA    JSR $CA35       Yes, get checksum f/ Buffer
E81E  A5 85       LDA $85         Compare checksum
E820  C5 87       CMP $87         with value reached
E822  F0 08       BEQ $E82C       Identical ?
E824  20 3E DE    JSR $DE3E       Get track and sector
E827  A9 50       LDA #$50        Error message
E829  20 45 E6    JSR $E645       "50 record not present"displayed
E82C[1] A5 F8     LDA $F8         Flag for get EOI (last char)
E82E  D0 A8       BNE $E7D8       Has last char. been used?
E830  68          PLA             Yes, repeat starting address
E831  85 89       STA $89         of program,,
E833  68          PLA             and put into
E834  85 88       STA $88         pointers $88/$89
E836  6C 88 00    JMP($0088)      Jump to prg via this pointer
----------------------------------------------------------------
[E7DC/E7E4/E7FA/E802]
Get byte from buffer
E839  20 35 CA    JSR $CA35       Get byte from sector
E83C  A5 F8       LDA $F8         Test EOI flag
E83E  D0 08       BNE $E848       Was that the last character?
E840  20 3E DE    JSR $DE3E       Yes, set track and sector
E843  A9 51       LDA #$51        Error message
E845  20 45 E6    JSR $E645       "51 Overflow in record"displayed
E848[1] A5 85     LDA $85         Get last character
E84A  60          RTS             Back to original routine
----------------------------------------------------------------
```

```
[E7E1/E7E9/E7FF/E809]
Implement checksum
E84B  18         CLC          Add new byte
E84C  65 87      ADC $87      to pre-existng vals ADDIEREN
E84E  69 00      ADC #$00     Calculate overflow
E850  85 87      STA $87      And note new checksum value
E852  60         RTS          Return from subroutine
-----------------------------------------------------------------
[9DC1]
Capture flag (ATN) from serial bus set
E853  AD 01 18   LDA $1801    [ERROR; Descr. in  7.1.3]
E856  A9 01      LDA #$01     Flag set for
E858  85 7C      STA $7C    "ATN Receive"
E85A   60        RTS          Return from subroutine
-----------------------------------------------------------------
[A7BD/EA56/EA68]
Routine/controlling serial bus
E85B  78         SEI          Disable bus/disk controller
E85C  A9 00      LDA #$00     Clear flags with zero:
E85E  85 7C      STA $7C      Set flags for "ATN RECEIVE"
E860  85 79      STA $79      Flag for listen
E862  85 7A      STA $7A      Flag for talk
E864  A2 45      LDX #$45     Set new
E866  9A         TXS          stack pointer
E867  A9 80      LDA #$80     Clear flags w/$80 (BIT7 active):
E869  85 F8      STA $F8      Flag / EOI (End of Transfer)
E86B  85 7D      STA $7D      Flag for ATN mode
E86D  20 B7 E9   JSR $E9B7    Clock set to high
E870  20 A5 E9   JSR $E9A5    Data lines set to low
E873  AD 00 18   LDA $1800    Get bus control register
E876  09 10      ORA #$10     ATN request cleared
E878  8D 00 18   STA $1800    and given on bus
E87B1 AD 00 18   LDA $1800    Bus status repeated
E87E  10 57      BPL $E8D7    Is ATN SET?
E880  29 04      AND #$04     No, mask clock line
E882  D0 F7      BNE $E87B    Is clock set ?
E8841 20 C9 E9   JSR $E9C9    Yes, readin commnd word from
E887  C9 3F      CMP #$3F     bus and compare with UNLIST
E889  D0 06      BNE $E891    Identical ?
E88B  A9 00      LDA #$00     Yes, clear flag for
E88D  85 79      STA $79      LISTEN
E88F  F0 71      BEQ $E902    Jump back to $E902
E8911 C9 5F      CMP #$5F     Compare with UNTALK
E893  D0 06      BNE $E89B    Identical ?
E895  A9 00      LDA #$00     Yes, clear flag for
E897  85 7A      STA $7A      TALK
E899  F0 67      BEQ $E902    Jump back to $E902
```

```
E89B¹ C5 78      CMP $78       Talk address label
E89D  D0 0A      BNE $E8A9     Should talk addr be recevng?
E89F  A9 01      LDA #$01      Yes, set flag for
E8A1  85 7A      STA $7A       TALK
E8A3  A9 00      LDA #$00      Flag for LISTEN
E8A5  85 79      STA $79       cleared
E8A7  F0 29      BEQ $E8D2     Jump back to $E8D2
E8A9¹ C5 77      CMP $77       LISTEN address label
E8AB  D0 0A      BNE $E8B7     Listen addr be receiving?
E8AD  A9 01      LDA #$01      Yes, set flag for
E8AF  85 79      STA $79       LISTEN
E8B1  A9 00      LDA #$00      Flag for TALK
E8B3  85 7A      STA $7A       cleared
E8B5  F0 1B      BEQ $E8D2     Jump back to $E8D2
E8B7  AA         TAX           Note command
E8B8  29 60      AND #$60      Isolate command bits
E8BA  C9 60      CMP #$60      for testing
E8BC  D0 3F      BNE $E8FD     Identical ?
E8BE  8A         TXA           Yes, repeat and note
E8BF  85 84      STA $84       command word
E8C1  29 0F      AND #$0F      Set up proper channel number
E8C3  85 83      STA $83       and save it
E8C5  A5 84      LDA $84       Repeat command word
E8C7  29 F0      AND #$F0      Combine address bits
E8C9  C9 E0      CMP #$E0      Compare with CLOSE command
E8CB  D0 35      BNE $E902     Identical ?
E8CD  58         CLI           Yes, enable disk controller
E8CE  20 C0 DA   JSR $DAC0     Close call
E8D1  78         SEI           Disable disk/bus controller
E8D2² 2C 00 18   BIT $1800     Check ATN bit
E8D5  30 AD      BMI $E884     ATN active?; if so, wait
E8D7³ A9 00      LDA #$00      No,
E8D9  85 7D      STA $7D       Clear flag for command mode
E8DB  AD 00 18   LDA $1800     Bus control register
E8DE  29 EF      AND #$EF      Clear ATN
E8E0  8D 00 18   STA $1800     and send over bus
E8E3  A5 79      LDA $79       Flag for LISTEN
E8E5  F0 06      BEQ $E8ED     Flag set?
E8E7  20 2E EA   JSR $EA2E     Data from bus put to buffer
E8EA  4C 6B 83   JMP $836B     Wait for next command word
E8ED¹ A5 7A      LDA $7A       Flag for TALK
E8EF  F0 09      BEQ $E8FA     active?
E8F1  20 9C E9   JSR $E99C     Data set high
E8F4  20 AE E9   JSR $E9AE     Clock set low
```

```
E8F7    20 09 E9    JSR $E909        Buffer data sent over bus
E8FA1   4C 4E EA    JMP $EA4E        Wait for next command word
E8FD1   A9 10       LDA #$10         No TALK or LIST commands
E8FF    8D 00 18    STA $1800        Data lines reset
E9024   2C 00 18    BIT $1800        Check ATN
E905    10 D0       BPL $E8D7        Is ATN reset?
E907    30 F9       BMI $E902        No, wait until command end
------------------------------------------------------------------
[E8F7]
Data sent after talk call
E909    78          SEI              Disable disk controller
E90A    20 EB D0    JSR $D0EB        Look for free channel and open
E90D    B0 06       BCS $E915        Is there a free channel?
E90F1   A6 82       LDX $82          Yes, get current channel number
E911    B5 F2       LDA $F2,X        and corresponding status
E913    30 01       BMI $E916        Is channel set to read?
E9151   60          RTS              No, return from subroutine
E9161   20 59 EA    JSR $EA59        ATN-Line test
E919    20 C0 E9    JSR $E9C0        Read value from bus register
E91C    29 01       AND #$01         and get data entry
E91E    08          PHP              Note state of data line
E91F    20 B7 E9    JSR $E9B7        Clock output set to low
E922    28          PLP              Get data line status again
E923    F0 12       BEQ $E937        Was data set ?
E9251   20 59 EA    JSR $EA59        Yes, test for ATN command mode
E928    20 C0 E9    JSR $E9C0        Get value from bus register
E92B    29 01       AND #$01         Isolate data line
E92D    D0 F6       BNE $E925        Wait until data is set to low
E92F    A6 82       LDX $82          Number of internal channels
E931    B5 F2       LDA $F2,X        Get appropriate status
E933    29 08       AND #$08         and test flag for EOI
E935    D0 14       BNE $E94B        Last character been sent?
E9372   20 59 EA    JSR $EA59        Yes, test for ATN mode
E93A    20 C0 E9    JSR $E9C0        Get value from bus register
E93D    29 01       AND #$01         and test data line
E93F    D0 F6       BNE $E937        Wait until data is low
E9411   20 59 EA    JSR $EA59        Test, for ATN command mode
E944    20 C0 E9    JSR $E9C0        Get value from bus register
E947    29 01       AND #$01         and isolate data line
E949    F0 F6       BEQ $E941        Wait until data input is high
E94B2   20 AE E9    JSR $E9AE        Clock output set high
E94E    20 59 EA    JSR $EA59        Test for ATN mode
E951    20 C0 E9    JSR $E9C0        Get value from bus register
E954    29 01       AND #$01         and analyze data
E956    D0 F3       BNE $E94B        Wait until data is set low
E958    A9 08       LDA #$08         Set number of bits per byte
E95A    85 98       STA $98          in counter
```

```
E95C¹ 20 C0 E9  JSR $E9C0      Get value from bus register
E95F  29 01     AND #$01       and test data line
E961  D0 36     BNE $E999      Is data low?
E963  A6 82     LDX $82        Yes, current channel number
E965  BD 3E 02  LDA $023E,X    Get corresponding data byte
E968  6A        ROR A          and save first bit in carry
E969  9D 3E 02  STA $023E,X    Note remainder
E96C  B0 05     BCS $E973      Is bit =1 ?
E96E  20 A5 E9  JSR $E9A5      No, data line is set high
E971  D0 03     BNE $E976      Jump back to $E976
E973¹ 20 9C E9  SR  $E99C      Data line set to low
E976¹ 20 B7 E9  JSR $E9B7      Clock line set to low
E979  A5 23     LDA $23        Test flag for bus mode
E97B  D0 03     BNE $E980      Is bus in 1540 mode ?
E97D  20 F3 FE  JSR $FEF3      No, 42-Cycle time delay
E980¹ 20 FB FE  JSR $FEFB      Data set low, Clock set high
E983  C6 98     DEC $98        Number of bits to be sent
E985  D0 D5     BNE $E95C      Is byte already sent?
E987¹ 20 59 EA  JSR $EA59      Yes, test for ATN mode
E98A  20 C0 E9  JSR $E9C0      Get value from bus register
E98D  29 01     AND #$01       and check data line
E98F  F0 F6     BEQ $E987      Is data set?
E991  58        CLI            Yes—Enable disk controller
E992  20 AA D3  JSR $D3AA      Get next data byte from Buffer
E995  78        SEI            Disable disk controller again
E996  4C 0F E9  JMP $E90F      and sent over bus
E999¹ 4C 4E EA  JMP $EA4E      Wait for next command
-------------------------------------------------------------------
[817B/8291/82D8/8300/E8F1/E973/E9D7/E9FA/FEFE]
Data line on low set
E99C  AD 00 18  LDA $1800      Read bus control register
E99F  29 FD     AND #$FD       Clear bit for
E9A1  8D 00 18  STA $1800      data line
E9A4  60        RTS            Return from subroutine
-------------------------------------------------------------------
[80E6/828C/82F8/833C/E870/E96E/E9F2/EA28]
Data line on high set
E9A5  AD 00 18  LDA $1800      Get bus control register
E9A8  09 02     ORA #$02       and set bit for
E9AA  8D 00 18  STA $1800      data line
E9AD  60        RTS            Return from subroutine
-------------------------------------------------------------------
```

```
Clock line set high
[817E/822C/E8F4/E94B/FEFB]
E9AE  AD 00 18  LDA $1800         Get bus control register
E9B1  09 08     ORA #$08          and set bit for clock
E9B3  8D 00 18  STA $1800         Line
E9B6  60        RTS               Return from subroutine
-------------------------------------------------------------
Clock line set low
[80E3/8200/829E/8438/E86D/E91F/E976]
E9B7  AD 00 18  LDA $1800         Get bus control register
E9BA  29 F7     AND #$F7          and clear bit for
E9BC  8D 00 18  STA $1800         clock line
E9BF  60        RTS               Return from subroutine
-------------------------------------------------------------
Values read from bus
[819C/81FA/8209/821B/8225/8232/827A/82B5/82D1/82EF/8306/8331/E919/E928]
[E93A/E944/E951/E95C/E98A/E9C6/E9D0/E9E9/EA00/EA1D]
E9C0  AD 00 18  LDA $1800         Get control register
E9C3 CD 00 18  CMP $1800          Get it again and compare with
E9C6  D0 F8     BNE $E9C0         last value; values constant?
E9C8  60        RTS               Yes, return from subroutine
-------------------------------------------------------------
Data received after listen call
[E884/EA44]
E9C9  A9 08     LDA #$08          Number of bits per data byte
E9CB  85 98     STA $98           Initialize counter
E9CD1 20 59 EA  JSR $EA59         ATN test
E9D0  20 C0 E9  JSR $E9C0         Get bus control register
E9D3  29 04     AND #$04          and test clock line
E9D5  D0 F6     BNE $E9CD         Clock active?
E9D7  20 9C E9  JSR $E99C         Yes, activate data line
E9DA  A9 01     LDA #$01          [ERROR—see 7.1.4]
E9DC  4C 20 FF  JMP $FF20         Wait til data  is low; timer set
E9DF1 20 59 EA  JSR $EA59         ATN test
E9E2  AD 0D 18  LDA $180D         Get interrupt flags
E9E5  29 40     AND #$40          and test flag for timer1
E9E7  D0 09     BNE $E9F2         Is timer running?
E9E9  20 C0 E9  JSR $E9C0         No, get value from bus register
E9EC  29 04     AND #$04          Isolate clock input
E9EE  F0 EF     BEQ $E9DF         Is clock set?
E9F0  D0 19     BNE $EA0B         Yes, jump back to $EA0B
E9F21 20 A5 E9  JSR $E9A5         Data line set high
E9F5  A2 0A     LDX #$0A          Delay counter set
E9F71 CA        DEX               Delay of 51 Cycles
E9F8  D0 FD     BNE $E9F7         Is delay running?
E9FA  20 9C E9  JSR $E99C         Yes, data set low
E9FD1 20 59 EA  JSR $EA59         Test bus for ATN command
```

```
EA00   20 C0 E9   JSR $E9C0         Get bus control register
EA03   29 04      AND #$04          Test clock line
EA05   F0 F6      BEQ $E9FD         Clock set?
EA07   A9 00      LDA #$00          Yes, end of file
EA09   85 F8      STA $F8           EOI flag set
EA0B3  AD 00 18   LDA $1800         Get control register again
EA0E   49 01      EOR #$01          Correct data bits of original
EA10   4A         LSR A             value, and put in carry
EA11   29 02      AND #$02          Test clock line (set by LSR)
EA13   D0 F6      BNE $EA0B         Clock set, file in order?
EA15   EA         NOP               Yes, empty register
EA16   EA         NOP               (Can't be used for your
EA17   EA         NOP               own data)
EA18   66 85      ROR $85           Move data bits into temp. buffer
EA1A1  20 59 EA   JSR $EA59         Test ATN
EA1D   20 C0 E9   JSR $E9C0         Read bus control register
EA20   29 04      AND #$04          Test clock line
EA22   F0 F6      BEQ $EA1A         Is clock set?
EA24   C6 98      DEC $98           Yes,counter set from # data bits
EA26   D0 E3      BNE $EA0B         8 bits read yet?
EA28   20 A5 E9   JSR $E9A5         Yes, set data line to low
EA2B   A5 85      LDA $85           Data byte taken
EA2D   60         RTS               Return from subroutine
-------------------------------------------------------------
[EA4B/E8E7]
Data taken from bus and put into current buffer
EA2E   78         SEI               Disable disk controller
EA2F   20 07 D1   JSR $D107         Write channel laid out
EA32   B0 05      BCS $EA39         Found a free channel?
EA34   B5 F2      LDA $F2,X         Yes, read proper channel status
EA36   6A         ROR A             Is the channel
EA37   B0 0B      BCS $EA44         active?
EA391  A5 84      LDA $84           Yes, get secondary address
EA3B   29 F0      AND #$F0          and isolate command bits
EA3D   C9 F0      CMP #$F0          Compare with OPEN command
EA3F   F0 03      BEQ $EA44         Identical?
EA41   4C 4E EA   JMP $EA4E         No, wait for next command
EA442  20 C9 E9   JSR $E9C9         Get byte from bus
EA47   58         CLI               Enable disk controller again
EA48   20 B7 CF   JSR $CFB7         Data byte transferred frm buffer
EA4B   4C 2E EA   JMP $EA2E         Get next byte
EA4E3  A9 00      LDA #$00          Bus control register
EA50   8D 00 18   STA $1800         Reset
EA53   4C 6B 83   JMP $836B         Wait for next command
-------------------------------------------------------------
Unused prg. area from 1541 DOS
-------------------------------------------------------------
```

```
Test if command has been transferred
[8199/81F7/8206/8218/8222/822F/82B2/82CE/82E5/8303/832E/8425/85F6]
[869D/8E14/E916/E925/E937/E941/E94E/E987/E9CD/E9DF/E9FD/EA1A]
EA59   A5 7D     LDA $7D           Flag for  "ATN active"
EA5B   F0 06     BEQ $EA63         ATN set ?
EA5D   AD 00 18  LDA $1800         Yes, current status of ATN line
EA60   10 09     BPL $EA6B         Is ATN also set?
EA62¹  60        RTS               Yes, return from subroutine
EA63¹  AD 00 18  LDA $1800         Get current ATN status
EA66   10 FA     BPL $EA62         Is ATN still set?
EA68   4C B3 A7  JMP $A7B3         Yes, get command from bus
EA6B¹  4C AC A9  JMP $A9AC         ATN reset
-----------------------------------------------------------------
[EAB5/EABE/EAC4]
Hardware error message (Infinite loop) —LED blinking
EA6E   A2 00     LDX #$00
EA70   2C A6 6F  .BYTE $2C         Jmp to next 2 bytes (bit command)
-----------------------------------------------------------------
[92B7/EB1F]
RAM or ROM error (TEST and CHECKSUM)
EA71   A6 6F     LDX $6F           Get blink counter
EA73   9A        TXS               and note it
EA74¹  BA        TSX               Get blink value again
EA75¹  A9 08     LDA #$08          LED bit (bit 3) set
EA77   0D 00 1C  ORA $1C00         in disk controller register and
EA7A   4C EA FE  JMP $FEEA         LED activated; more at $EA7D
EA7D   98        TYA               (0) counter routine
EA7E¹  18        CLC               initialized
EA7F¹  69 01     ADC #$01          Delay counter
EA81   D0 FC     BNE $EA7F         for approximately 0.3 / 0.1 sec
EA83   88        DEY               after 1 or 2 Mhz time
EA84   D0 F8     BNE $EA7E         Is time running?
EA86   AD 00 1C  LDA $1C00         Yes, get control register
EA89   29 F7     AND #$F7          and combine bit for
EA8B   8D 00 1C  STA $1C00         "LED on"
EA8E¹  98        TYA               (0)  counter routine
EA8F¹  18        CLC               intialized
EA90¹  69 01     ADC #$01          Delay counter
EA92   D0 FC     BNE $EA90         For approximately 0.3 / 0.1 sec
EA94   88        DEY               after 1 OR 2 Mhz time
EA95   D0 F8     BNE $EA8F         Is timer on?
EA97   CA        DEX               Blink counter
EA98   10 DB     BPL $EA75         still blinking?
EA9A   E0 FC     CPX #$FC          No, wait approximately 1/0.5 sec
EA9C   D0 F0     BNE $EA8E         Time running?
EA9E   F0 D4     BEQ $EA74         Yes, blink again before starting
-----------------------------------------------------------------
```

```
[89EC/Jump over system vector/ FFFC]
1571 Reset jump
-------------------------------------------------------------
RAM and ROM test
EAA0  78         SEI          Disable bus/disk controller
EAA1  D8         CLD          Arithmetic mode set for binary
EAA2  A2 66      LDX #$66     Value for DDRA
EAA4  4C 10 FF   JMP $FF10    VIA'S initialized; jump to $EAA7
EAA7  E8         INX          [ERROR—see 7.1.4]
EAA8  A0 00      LDY #$00     Initialize offset counter
EAAA  A2 00      LDX #$00     Set pointer to zeropage area
EAAC1 8A         TXA          Write number of memory cells
EAAD  95 00      STA $00,X    Into memory cells
EAAF  E8         INX          Pick next memory cell
EAB0  D0 FA      BNE $EAAC    $100 address reached?
EAB21 8A         TXA          Yes, compare value of memory
EAB3  D5 00      CMP $00,X    location with this value
EAB5  D0 B7      BNE $EA6E    Both identical?
EAB71 F6 00      INC $00,X    Yes, increase memory location
EAB9  C8         INY          Offset  mem address contents—set
EABA  D0 FB      BNE $EAB7    All values tested set?
EABC  D5 00      CMP $00,X    Yes, comp with memory
EABE  D0 AE      BNE $EA6E    Identical?
EAC0  94 00      STY $00,X    Yes, clear memory locatn with 0
EAC2  B5 00      LDA $00,X    Get contents again
EAC4  D0 A8      BNE $EA6E    Was clear in order?
EAC6  E8         INX          Yes, pick next memory location
EAC7  D0 E9      BNE $EAB2    Reached $100 yet?
EAC9  E6 6F      INC $6F      Yes, incremt error blink counter
EACB  A2 80      LDX #$80     Starting address of
EACD  86 76      STX $76      Operating system ROM
EACF  A9 00      LDA #$00     Set to $8000 in pointers $75/76
EAD1  85 75      STA $75      Set
EAD3  A0 02      LDY #$02     Ignore checksum bytes
EAD5  18         CLC          ROM pointer set
EAD61 E6 76      INC $76      To next memory page
EAD81 71 75      ADC ($75),Y  ROM value for CHECKSUM
EADA  C8         INY          Pointer turned to next byte
EADB  D0 FB      BNE $EAD8    Whole memory page considered?
EADD  CA         DEX          Yes, # of ROM memory pages
EADE  D0 F6      BNE $EAD6    Entire ROM checked?
EAE0  69 FF      ADC #$FF     Yes, calculate checksum value
EAE2  85 76      STA $76      and note result
EAE4  D0 39      BNE $EB1F    Is error on hand?
EAE6  EA         NOP          No, empty space
EAE7  EA         NOP          resulting from modification
EAE8  EA         NOP          of 1541
```

```
EAE9  EA        NOP               ROM
EAEA  A9 01     LDA #$01          Pick
EAEC  85 76     STA $76           memory page 1
EAEE  E6 6F     INC $6F           Page # set in blink counter
EAF0  A2 07     LDX #$07          Number of RAM pages
EAF2² 98        TYA               Clear value(Memory #)
EAF3  18        CLC               Compute # of
EAF4  65 76     ADC $76           memory pages and
EAF6  91 75     STA ($75),Y       write to memory
EAF8  C8        INY               Set pointer to next byte
EAF9  D0 F7     BNE $EAF2         Entire memory page cleared?
EAFB  E6 76     INC $76           Yes, set pointer to next page
EAFD  CA        DEX               Number of RAM pages
EAFE  D0 F2     BNE $EAF2         Whole RAM cleared already?
EB00  A2 07     LDX #$07          Yes, # of RAM pages
EB02¹ C6 76     DEC $76           RAM pointer to preceding page
EB04¹ 88        DEY               Number of pages yet to be tested
EB05  98        TYA               Get position #
EB06  18        CLC               and
EB07  65 76     ADC $76           calculate page #—
EB09  D1 75     CMP ($75),Y       compare with clear value
EB0B  D0 12     BNE $EB1F         Is memory location right?
EB0D  49 FF     EOR #$FF          Yes, change values of all bits
EB0F  91 75     STA ($75),Y       And test for other valences
EB11  51 75     EOR ($75),Y       Test result
EB13  91 75     STA ($75),Y       and clear memory cell (0)
EB15  D0 08     BNE $EB1F         Was test successful?
EB17  98        TYA               Yes, set processr flgs fr Y-value
EB18  D0 EA     BNE $EB04         End of memory page?
EB1A  CA        DEX               Yes, pick next page
EB1B  D0 E5     BNE $EB02         Entire RAM tested yet?
EB1D  F0 03     BEQ $EB22         Yes, jump to $EB22
EB1F³ 4C 71 EA  JMP $EA71         Hardware error display
----------------------------------------------------------------
[EB1D/EB25:A7C4]
Initialize zeropage
EB22 4C C0 A7 JMP $A7C0           Stack set to $0100-$0145
EB25  AD 00 1C LDA $1C00          Get disk drive control register
EB28  29 F7    AND #$F7           and switch off disk
EB2A  8D 00 1C STA $1C00          Drive LED
EB2D  A9 01    LDA #$03           CA1 (ATN) triggered positive &
EB2F  8D 0C 18 STA $180C          CA2 (WP) to negative
EB32  A9 82    LDA #$82           "Interupt from CA1 active"
EB34  8D 0D 18 STA $180D          Flag cleared
EB37  8D 0E 18 STA $180E          and activated
EB3A  AD 00 18 LDA $1800          Hardwr-dependent determination
EB3D  29 60    AND #$60           of device address
```

```
EB3F   0A         ASL A         gotten, and the two signifi-
EB40   2A         ROL A         cant bits 5 and 6
EB41   2A         ROL A         shifted to positions
EB42   2A         ROL A         0 and 1
EB43   09 48      ORA #$48      Device # for talker operation
EB45   85 78      STA $78       generated and stored
EB47   49 60      EOR #$60      Device # for listener operation
EB49   85 77      STA $77       created and set
EB4B   A2 00      LDX #$00      pointer to buffer pointer
EB4D   A0 00      LDY #$00      High byte table pointer
EB4F1  A9 00      LDA #$00      Low byte value
EB51   95 99      STA $99,X     Clear buffer pointer low byte
EB53   E8         INX           Set high byte pointer
EB54   B9 E0 FE   LDA $FEE0,Y   Get buffer address (high byte)
EB57   95 99      STA $99,X     and put in pointer
EB59   E8         INX           Pointer to next buffer pointer
EB5A   C8         INY           Pointer to next high byte
EB5B   C0 05      CPY #$05      Buffer #
EB5D   D0 F0      BNE $EB4F     All buffer addresses laid out?
EB5F   A9 00      LDA #$00      Yes, low byte of input buffer
EB61   95 99      STA $99,X     pointer
EB63   E8         INX           High byte pointer
EB64   A9 02      LDA #$02      Turn buffer pointer to
EB66   95 99      STA $99,X     address $200
EB68   E8         INX           Pointer to next byte
EB69   A9 D5      LDA #$D5      Low byte of error buffer
EB6B   95 99      STA $99,X     set
EB6D   E8         INX           Turn pointer to next byte
EB6E   A9 02      LDA #$02      Error message buffer turned to
EB70   95 99      STA $99,X     address $02D5
EB72   A9 FF      LDA #$FF      "Channel free" value
EB74   A2 12      LDX #$12      No. of secondary addresses (19)
EB761  9D 2B 02   STA $022B,X   Free channel
EB79   CA         DEX           Next secondary address
EB7A   10 FA      BPL $EB76     Entire table used up?
EB7C   A2 05      LDX #$05      Yes, # of internal channels (6)
EB7E1  95 A7      STA $A7,X     1. Buffer freed
EB80   95 AE      STA $AE,X     2. Buffer freed
EB82   95 CD      STA $CD,X     3. Buffer freed
EB84   CA         DEX           Set next channel
EB85   10 F7      BPL $EB7E     All channels considered?
EB87   A9 05      LDA #$05      Yes, assign input buffer
EB89   85 AB      STA $AB       to channel 4
EB8B   A9 06      LDA #$06      Assign ERROR buffer
EB8D   85 AC      STA $AC       to channel 5
EB8F   A9 FF      LDA #$FF      Value/"No buffer allotted"
EB91   85 AD      STA $AD       in channel 6 (1st buffer
```

```
EB93  85 B4     STA $B4         and 2nd buffer)
EB95  A9 05     LDA #$05        Secondary address 16 leads
EB97  8D 3B 02  STA $023B       to channel 5
EB9A  A9 84     LDA #$84        Secondary address 15 to channel 4
EB9C  8D 3A 02  STA $023A       (Status:WRITE channel only)
EB9F  A9 0F     LDA #$0F        Flags for channel layout arranged
EBA1  8D 56 02  STA $0256       Channels 0-3 freed up
EBA4  A9 01     LDA #$01        Flag for "Write channel"
EBA6  85 F6     STA $F6         Set for channel 4
EBA8  A9 88     LDA #$88        Flag/"Read channel/no EOI"
EBAA  85 F7     STA $F7         Set for channel 5
EBAC  A9 E0     LDA #$E0        Flags set to buffer layout
EBAE  8D 4F 02  STA $024F       (Set bit=Set buffer)
EBB1  A9 FF     LDA #$FF        Buffers 0-4
EBB3  8D 50 02  STA $0250       freed up
EBB6  A9 01     LDA #$01        Write-protect notch
EBB8  85 1C     STA $1C         status flag
EBBA  85 1D     STA $1D         cleared
EBBC  20 63 CB  JSR $CB63       Jmp table pointer/Switch commands
EBBF  20 FA CE  JSR $CEFA       Initialize buffer channel table
EBC2  20 82 FF  JSR $FF82       Activate disk controller routine
EBC5  A9 22     LDA #$22        Pointer to NMI or SWITCH
EBC7  85 65     STA $65         command between 1541 and 1540
EBC9  A9 EB     LDA #$EB        in $65/$66 set
EBCB  85 66     STA $66         in $EB22
EBCD  A9 06     LDA #$06        Sector pawning (6)
EBCF  85 69     STA $69         determined
EBD1  A9 05     LDA #$05        Number of reader searches by
EBD3  85 6A     STA $6A         error set to 5
EBD5  A9 73     LDA #$73        DOS power-up message
EBD7  20 C1 E6  JSR $E6C1       "73 CBM DOS V3.0 1571"displayed
EBDA  A9 00     LDA #$00        Bus lines
EBDC  8D 00 18  STA $1800       reset
EBDF  A9 1A     LDA #$1A        Input/output layout %00011010
EBE1  8D 02 18  STA $1802       determined
EBE4  20 86 A7  JSR $A786       CIA 6526 initialized
-------------------------------------------------------------
[836E/E8EA/EA53]
Wait loop for command recognition
EBE7  58        CLI             Enable bus/disk controller
EBE8  AD 00 18  LDA $1800       Get bus control register
EBEB  29 E5     AND #$E5        and clear all outputs
EBED  8D 00 18  STA $1800       and set bus to output status
EBF0  AD 55 02  LDA $0255       Flag/"Command received"
EBF3  F0 0A     BEQ $EBFF       set?
EBF5  A9 00     LDA #$00        Yes, command flag
EBF7  8D 55 02  STA $0255       cleared
```

```
EBFA  EA          NOP              NOP from modification
EBFB  EA          NOP              of 1541 ROM
EBFC  4C 1C A6    JMP $A61C        Command from COMP. executed
-----------------------------------------------------------------
[A628/EBF3/EC9B]
Wait for command
EBFF  58          CLI              Enable bus/disk controller
EC00  A5 7C       LDA $7C          flag for ATN receive
EC02  F0 03       BEQ $EC07        set?
EC04  4C 94 A6    JMP $A694        Yes-
EC07¹ 58          CLI              Enable disk/bus controller again
EC08  A9 0E       LDA #$0E         Largest secondary address for
EC0A  85 72       STA $72          files available
EC0C  A9 00       LDA #$00         Outstanding job counter
EC0E  85 6F       STA $6F          cleared for disk drive 0 and
EC10  85 70       STA $70          disk drive 1
EC12¹ A6 72       LDX $72          Get secondary address
EC14  BD 2B 02    LDA $022B,X      and check corresponding channel
EC17  C9 FF       CMP #$FF         status "free" value
EC19  F0 10       BEQ $EC2B        Is channel free?
EC1B  29 3F       AND #$3F         No, get and note # of in-
EC1D  85 82       STA $82          ternal channels allotted
EC1F  20 93 DF    JSR $DF93        Get and note
EC22  AA          TAX              buffer #
EC23  BD 5B 02    LDA $025B,X      Job code of buffer determined
EC26  29 01       AND #$01         and last-used disk drive
EC28  AA          TAX              noted
EC29  F6 6F       INC $6F,X        Increment # of jobs
EC2B¹ C6 72       DEC $72          Next secondary address
EC2D  10 E3       BPL $EC12        All channels checked?
EC2F  A0 04       LDY #$04         Yes, buffer #
EC31¹ B9 00 00    LDA $0000,Y      Get buffer job code
EC34  10 05       BPL $EC3B        Is job in process?
EC36  29 01       AND #$01         Yes, get and note disk drive
EC38  AA          TAX              number
EC39  F6 6F       INC $6F,X        Increment # of jobs
EC3B¹ 88          DEY              Choose next buffer
EC3C  10 F3       BPL $EC31        All jobs tested?
EC3E  78          SEI              YES-Disable bus/disk controller
EC3F  AD 00 1C    LDA $1C00        Get disk control register
EC42  29 F7       AND #$F7         LED reset and mask generated
EC44  48          PHA              For "LED OUT" noted
EC45  A5 7F       LDA $7F          Current drive
EC47  85 86       STA $86          Note current disk
EC49  A9 00       LDA #$00         drive 0
EC4B  85 7F       STA $7F          chosen
EC4D  A5 6F       LDA $6F          Number of jobs for drive 0
```

```
EC4F  F0 0B     BEQ $EC5C          Are any jobs accomplished?
EC51  A5 1C     LDA $1C            Yes, flag for write-protect light
EC53  F0 03     BEQ $EC58          Should diskette be initialized?
EC55  20 13 D3  JSR $D313          Diskette initialization
EC58¹ 68        PLA                Get mask for drive control again,
EC59  09 08     ORA #$08           Switch LED and save
EC5B  48        PHA                again
EC5C¹ E6 7F     INC $7F            Choose drive 1
EC5E  A5 70     LDA $70            Drive 1 job counter
EC60  F0 0B     BEQ $EC6D          Are drive 1 jobs done?
EC62  A5 1D     LDA $1D            Yes,write-protect flag for drive1
EC64  F0 03     BEQ $EC69          Diskette change found?
EC66  20 13 D3  JSR $D313          Close all drive channels
EC69¹ 68        PLA                Bring back drive control mask
EC6A  09 00     ORA #$00           and set LED for drive 1
EC6C  48        PHA                -note again
EC6D¹ A5 86     LDA $86            Call back and take up
EC6F  85 7F     STA $7F            Current disk drive
EC71  68        PLA                Get disk control mask
------------------------------------------------------------
LED error blinking control
EC72  AE 6C 02  LDX $026C          Test error flag
EC75  F0 21     BEQ $EC98          Set?
EC77  AD 00 1C  LDA $1C00          Yes, get control register
EC7A  E0 80     CPX #$80           Test blink phase counter
EC7C  D0 03     BNE $EC81          timer reset?
EC7E  4C 8B EC  JMP $EC8B          No, go on
EC81¹ AE 05 18  LDX $1805          High byte of timer 1
EC84  30 12     BMI $EC98          Is counter running?
EC86  A2 A0     LDX #$A0           Yes, high byte
EC88  8E 05 18  STX $1805          reset
EC8B¹ CE 6C 02  DEC $026C          Blink counter decremented
EC8E  D0 08     BNE $EC98          Is counter running?
EC90  4D 6D 02  EOR $026D          LED switched
EC93  A2 10     LDX #$10           Blink counter 0.4/ 0.2 sec.
EC95  8E 6C 02  STX $026C          Delay set
EC98³ 8D 00 1C  STA $1C00          LED controlled
EC9B  4C FF EB  JMP $EBFF          Blink some more
------------------------------------------------------------
```

```
[DAA7]
Directory for 'Load "$"'sets up directory to load as BASIC program
EC9E  A9 00     LDA #$00        Set zero as current
ECA0  85 83     STA $83         2ndary address(load channel)
ECA2  A9 01     LDA #$01        Look for read channel and
ECA4  20 E2 D1  JSR $D1E2       set pointer position in appro-
ECA7  A9 00     LDA #$00        priate buffer
ECA9  20 C8 D4  JSR $D4C8       Reset buffer to null
ECAC  A6 82     LDX $82         Number of channels found
ECAE  A9 00     LDA #$00        Clear pointer to end of buffer
ECB0  9D 44 02  STA $0244,X     entrance
ECB3  20 93 DF  JSR $DF93       Get and note number of
ECB6  AA        TAX             buffers chosen
ECB7  A5 7F     LDA $7F         Get current drive # and
ECB9  9D 5B 02  STA $025B,X     arrange drive table buffer
ECBC  A9 01     LDA #$01        starting address of "Imagi
ECBE  20 F1 CF  JSR $CFF1       ary" BASIC program ($0401)
ECC1  A9 04     LDA #$04        written to
ECC3  20 F1 CF  JSR $CFF1       current buffer
ECC6  A9 01     LDA #$01        Two fillbytes as placeholder
ECC8  20 F1 CF  JSR $CFF1       for the bASIC line pointer to
ECCB  20 F1 CF  JSR $CFF1       write to the buffer
ECCE  AD 72 02  LDA $0272       Get drive number, and put in
ECD1  20 F1 CF  JSR $CFF1       buffer as low byte of the line
ECD4  A9 00     LDA #$00        number; the high byte
ECD6  20 F1 CF  JSR $CFF1       is set to null
ECD9  20 59 ED  JSR $ED59       Transfer disk name to buffer
ECDC  20 93 DF  JSR $DF93       Get the number of the current
ECDF  0A        ASL A           buffer, double, and
ECE0  AA        TAX             decremt pointer from current
ECE1  D6 99     DEC $99,X       buffer position by two
ECE3  D6 99     DEC $99,X       characters
ECE5  A9 00     LDA #$00        Store end-of-BASIC line
ECE7  20 F1 CF  JSR $CFF1       in buffer
ECEA  A9 01     LDA #$01        Set up two bytes as placeholders
ECEC  20 F1 CF  JSR $CFF1       for chaining of
ECEF  20 F1 CF  JSR $CFF1       BASIC lines
ECF2  20 CE C6  JSR $C6CE       Read entry from directory
ECF5  90 2C     BCC $ED23       All entries handled?
ECF7  AD 72 02  LDA $0272       No, # of blocks laid out (low)
ECFA  20 F1 CF  JSR $CFF1       as low byte of BASIC line # &
ECFD  AD 73 02  LDA $0273       high byte of # blocks as hi byte
ED00  20 F1 CF  JSR $CFF1       of BASIC line # in buffer
ED03  20 59 ED  JSR $ED59       Copy dir. entry in buffer
ED06  A9 00     LDA #$00        Set"End-Of-BASIC line"'
ED08  20 F1 CF  JSR $CFF1       in buffer
ED0B  D0 DD     BNE $ECEA       Buffer full?
```

```
ED0D[1] 20 93 DF JSR $DF93        No, get # of current buffers
ED10   0A       ASL A             Double number
ED11   AA       TAX               and reset pointer to current
ED12   A9 00    LDA #$00          position of corresponding
ED14   95 99    STA $99,X         buffer
ED16   A9 88    LDA #$88          Flag/"Directory not in buffer"
ED18   A4 82    LDY $82           Get channel number
ED1A   8D 54 02 STA $0254         Set flag
ED1D   99 F2 00 STA $00F2,Y       Switch channel status to read
ED20   A5 85    LDA $85           Get current data byte
ED22   60       RTS               Return from subroutine
-----------------------------------------------------------------
[ECF5]
Directory output ended
ED23   AD 72 02 LDA $0272         Take # of free blocks in
ED26   20 F1 CF JSR $CFF1         $0272/0273 as BASIC line #
ED29   AD 73 02 LDA $0273         and write to
ED2C   20 F1 CF JSR $CFF1         buffer
ED2F   20 59 ED JSR $ED59         Write"BLOCKS FREE"/ buffer
ED32   20 93 DF JSR $DF93         Get # of current buffer
ED35   0A       ASL A             Double number
ED36   AA       TAX               Pointer for current character
ED37   D6 99    DEC $99,X         position in buffer set in two
ED39   D6 99    DEC $99,X         bytes
ED3B   A9 00    LDA #$00          End-of-line marker and two
ED3D   20 F1 CF JSR $CFF1         blank string bytes(End-of-
ED40   20 F1 CF JSR $CFF1         program markers), put into
ED43   20 F1 CF JSR $CFF1         current buffer
ED46   20 93 DF JSR $DF93         Get current buffer number
ED49   0A       ASL A             Double number
ED4A   A8       TAY               Get number of bytes still
ED4B   B9 99 00 LDA $0099,Y       in buffer and set as pointer
ED4E   A6 82    LDX $82           for the end bytes transferred
ED50   9D 44 02 STA $0244,X       from
ED53   DE 44 02 DEC $0244,X       buffer
ED56   4C 0D ED  JMP $ED0D        End
-----------------------------------------------------------------
[ECD9/ED03/ED2F]
Copy directory entry into current buffer
ED59   A0 00    LDY #$00          Initialize buffer pointer
ED5B   B9 B1 02 LDA $02B1,Y       Get char. from directory buffer
ED5E   20 F1 CF JSR $CFF1         and transfer to current buffer
ED61   C8       INY               Set pointer to next character
ED62   C0 1B    CPY #$1B          Number of char. per entry
ED64   D0 F5    BNE $ED5B         All characters copied?
ED66   60       RTS               Yes, return from subroutine
-----------------------------------------------------------------
```

```
[D40E]
Get byte from directory
ED67   20 37 D1 JSR $D137         Get byte from file
ED6A   F0 01    BEQ $ED6D         End of file reached?
ED6C   60       RTS               No, return from subroutine
ED6D1  85 85    STA $85           Save last data byte
ED6F   A4 82    LDY $82           Get # of channels
ED71   B9 44 02 LDA $0244,Y       No. of bytes to be transferred
ED74   F0 08    BEQ $ED7E         No more data?
ED76   A9 80    LDA #$80          No, set channel status to
ED78   99 F2 00 STA $00F2,Y       "READ/EOI" and
ED7B   A5 85    LDA $85           get last data byte again
ED7D   60       RTS               Return from subroutine
ED7E1  48       PHA               Get # of data bytes
ED7F   20 EA EC JSR $ECEA         Produce directory line
ED82   68       PLA               Get last data byte again
ED83   60       RTS               Return from subroutine
---------------------------------------------------------------
[Jump to routine C146]
Routine for validate command
ED84   20 D1 C1 JSR $C1D1         Get drive # from command string
ED87   20 42 D0 JSR $D042         Initialize diskette
ED8A   A9 40    LDA #$40          Flag for"ILLEGAL BAM"'
ED8C   8D F9 02 STA $02F9         set
ED8F   20 C7 A7 JSR $A7C7         Produce new BAM
ED92   A9 00    LDA #$00          Clear pointer for directory
ED94   8D 92 02 STA $0292         entry; set search flag
ED97   20 AC C5 JSR $C5AC         Look for files in directory
ED9A   D0 3D    BNE $EDD9         Found an entry?
ED9C1  A9 00    LDA #$00          No, set sector numbers
ED9E   85 81    STA $81           to null
EDA0   AD 85 FE LDA $FE85         Get and set up
EDA3   85 80    STA $80           directory track number (18)
EDA5   20 E5 ED JSR $EDE5         Put directory track in BAM
EDA8   A9 00    LDA #$00          Clear flag for "ILLEGAL
EDAA   8D F9 02 STA $02F9         BAM"
EDAD   20 FF EE JSR $EEFF         Write BAM to diskette
EDB0   4C 94 C1 JMP $C194         "OK"displayed,end of command
---------------------------------------------------------------
[EDDD]
All blocks of a file put into BAM
EDB3   C8       INY               Set dir. buffer pointer to track
EDB4   B1 94    LDA ($94),Y       of first data block—and
EDB6   48       PHA               get (and note) track number
EDB7   C8       INY               Set buffer pointer to next char.
EDB8   B1 94    LDA ($94),Y       Get and save sector number
EDBA   48       PHA               of first data block
```

```
EDBB   A0 13     LDY #$13          Set buffer pointr to position of
EDBD   B1 94     LDA ($94),Y       side sector pointer & get track
EDBF   F0 0A     BEQ $EDCB         Side sector block avail.?
EDC1   85 80     ST $80            Yes, Save track number of first
EDC3   C8        INY               sector,get the sector # from
EDC4   B1 94     LDA ($94),Y       the directory
EDC6   85 81     STA $81           buffer
EDC8   20 E5 ED  JSR $EDE5         Read,lay out side sector blocks
EDCB1  68        PLA               Get sector number again and
EDCC   85 81     STA $81           save it
EDCE   68        PLA               Set up and save track
EDCF   85 80     STA $80           number again
EDD1   20 E5 ED  JSR $EDE5         Read data block, put into BAM
EDD41  20 04 C6  JSR $C604         Get next legal file entry
EDD7   F0 C3     BEQ $ED9C         All files checked?
EDD91  A0 00     LDY #$00          Set buffer pointer to 1st char.
EDDB   B1 94     LDA ($94),Y       Get identifier for filetype
EDDD   30 D4     BMI $EDB3         Are files closed properly?
EDDF   20 B6 C8  JSR $C8B6         No, clear file
EDE2   4C D4 ED  JMP $EDD4         Go on to next filename
-------------------------------------------------------------------
[EDA5/EDC8/EDD1]
All blocks following a file
EDE5   20 5F D5  JSR $D55F         Check track and sector number
EDE8   20 90 EF  JSR $EF90         Put current block in BAM
EDEB   20 75 D4  JSR $D475         Read block in buffer
EDEE1  A9 00     LDA #$00          Set buffer pointer to beginning
EDF0   20 C8 D4  JSR $D4C8         of file block
EDF3   20 37 D1  JSR $D137         Get 1st byte from file block and
EDF6   85 80     STA $80           save track of next block
EDF8   20 37 D1  JSR $D137         Get 2nd byte of file block and
EDFB   85 81     STA $81           store corresponding sector
EDFD   A5 80     LDA $80           Test track identifier for EOF;
EDFF   D0 03     BNE $EE04         last block of file?
EE01   4C 27 D2  JMP $D227         Yes, close channel and end
EE041  20 90 E   JSR $EF90         Block in BAM as stored identifir
EE07   20 4D D4  JSR $D44D         Read next file block and
EE0A   4C EE ED  JMP $EDEE         continue testing
-------------------------------------------------------------------
[Jump to routine C146]
Routine for new command
EE0D   20 12 C3  JSR $C312         Get drive # from command
EE10   A5 E2     LDA $E2           Get number
EE12   10 05     BPL $EE19         Number in order?
EE14   A9 33     LDA #$33          No, ERROR message
EE16   4C C8 C1  JMP $C1C8         "33 SYNTAX ERROR" output
EE191  29 01     AND #$01          Set up drive number and save
```

```
EE1B   85 7F      STA $7F          as current drive
EE1D   20 9C FF   JSR $FF9C        Set drive status and operate LED
EE20   A5 7F      LDA $7F          Number of current drive
EE22   0A         ASL A            and Double
EE23   AA         TAX              (2-byte-table)
EE24   AC 7B 02   LDY $027B        Compare position of ID pointer
EE27   CC 74 02   CPY $0274        with length of command string
EE2A   F0 1A      BEQ $EE46        Is a new ID given?
EE2C   B9 00 02   LDA $0200,Y      Yes,get and convey 1st ID char.
EE2F   95 12      STA $12,X        from input buffer
EE31   B9 01 02   LDA $0201,Y      Get the second ID
EE34   95 13      STA $13,X        character
EE36   20 07 D3   JSR $D307        Close all channels
EE39   A9 01      LDA #$01         Set first track
EE3B   85 80      STA $80          to be formatted
EE3D   20 2F FF   JSR $FF2F        format diskette
EE40   4C 64 A7   JMP $A764        Clear buffer for BAM
EE43   4C 56 EE   JMP $EE56        Make sector 18, 0
EE46[1] 20 42 D0  JSR $D042        initialize diskette
EE49   A6 7F      LDX $7F          Get current drive, and
EE4B   BD 01 01   LDA $0101,X      determine format identifiers to
EE4E   CD D5 FE   CMP $FED5        be read
EE51   F0 03      BEQ $EE56        Right format?
EE53   4C 72 D5   JMP $D572        No, output "POWER-ON"message
EE56[2] 20 C7 A7  JSR $A7C7        Create new BAM
EE59   A5 F9      LDA $F9          Current buffer number
EE5B   A8         TAY              Double number
EE5C   0A         ASL A            (Buffer pointer presented
EE5D   AA         TAX              as a 2-byte number)
EE5E   AD 88 FE   LDA $FE88        Get name position in sector 18,0
EE61   95 99      STA $99,X        and put in buffer pointer
EE63   AE 7A 02   LDX $027A        Get buffer number
EE66   A9 1B      LDA #$1B         Length of diskette name
EE68   20 6E C6   JSR $C66E        Copy disk name to BAM-buffer
EE6B   A0 12      LDY #$12         Diskette name pointer
EE6D   A6 7F      LDX $7F          Get current drive
EE6F   AD D5 FE   LDA $FED5        Get and store identifier for
EE72   9D 01 01   STA $0101,X      1541/1571 format
EE75   8A         TXA              Get and double
EE76   0A         ASL A            drive
EE77   AA         TAX              number
EE78   B5 12      LDA $12,X        Get drive ID and put
EE7A   91 94      STA ($94),Y      into buffer
EE7C   C8         INY              Buffer pointer set to next byte
EE7D   B5 13      LDA $13,X        Get 2 ID chars and
EE7F   91 94      STA ($94),Y      send to buffer
```

```
EE81  C8        INY               Buffer pointer set up for two
EE82  C8        INY               characters
EE83  A9 32     LDA #$32          "2A" written
EE85  91 94     STA ($94),Y       as identifier for format
EE87  C8        INY               in directory line with
EE88  AD D5 FE  LDA $FED5         Diskette name
EE8B  91 94     STA ($94),Y       and ID
EE8D  A0 02     LDY #$02          Write track number
EE8F  91 6D     STA ($6D),Y       in BAM
EE91  AD 85 FE  LDA $FE85         Set number of directory
EE94  85 80     STA $80           track
EE96  20 93 EF  JSR $EF93         Set BAM block in BAM as proof
EE99  A9 01     LDA #$01          Determine number of first
EE9B  85 81     STA $81           directory block
EE9D  20 93 EF  JSR $EF93         Put directory block in BAM
EEA0  20 FF EE  JSR $EEFF         Write new BAM to disk
EEA3  20 05 F0  JSR $F005         Clear BAM buffer
EEA6  A0 01     LDY #$01          Set buff pointer to2nd char.
EEA8  A9 FF     LDA #$FF          Write # of valid buffer bytes
EEAA  91 6D     STA ($6D),Y       to directory block
EEAC  20 64 D4  JSR $D464         Write directory block 18,1
EEAF  C6 81     DEC $81           Current sector # to null
EEB1  20 42 D0  JSR $D042         and read sector
EEB4  4C 94 C1  JMP $C194         "OK" message
--------------------------------------------------------------------
[A6C4/A708]
New 1541 BAM
EEB7  20 D1 F0  JSR $F0D1         Clear BAM buffer
EEBA  A0 00     LDY #$00          Initialize buffer pointer
EEBC  A9 12     LDA #$12          Move pointer to track # of next
EEBE  91 6D     STA ($6D),Y       block of track 18
EEC0  C8        INY               Set buffer pointer to sector #
EEC1  98        TYA               (1)
EEC2  91 6D     STA ($6D),Y       Sector number 1 taken
EEC4  C8        INY               Current buffer pointer
EEC5  C8        INY               moved further back by three
EEC6  C8        INY               characters
EEC7[1] A9 00   LDA #$00          Temporary storage for
EEC9  85 6F     STA $6F           list
EECB  85 70     STA $70           of blocks used
EECD  85 71     STA $71           Clear
EECF  98        TYA               Track number determined,
EED0  4A        LSR A             for working with the block
EED1  4A        LSR A             Availability map (BAM)
EED2  20 4B F2  JSR $F24B         Max. # of sectors determined
EED5  91 6D     STA ($6D),Y       and put into BAM
EED7  C8        INY               Buffer pointer to next byte
```

```
EED8   AA         TAX               Set counter for # of sectors
EED9¹  38         SEC               bitflag for "sector used"set
EEDA   26 6F      ROL $6F           Bit in 24-bit temp. storage
EEDC   26 70      ROL $70           Reserved blocks of track
EEDE   26 71      ROL $71           laid out
EEE0   CA         DEX               Set next sector in BAM
EEE1   D0 F6      BNE $EED9         All sectors of track?
EEE3¹  B5 6F      LDA $6F,X         Yes, write contents of temp.
EEE5   91 6D      STA ($6D),Y       memory into BAM buffer
EEE7   C8         INY               Buffer pointer to next byte
EEE8   E8         INX               Counter for # of temp.memory
EEE9   E0 03      CPX #$03          compared to three
EEEB   90 F6      BCC $EEE3         All temp. mem. bytes copied?
EEED   C0 90      CPY #$90          Yes, comp buff pointer w/ $90
EEEF   90 D6      BCC $EEC7         BAM bits of all tracks dtrmnd?
EEF1   4C 75 D0   JMP $D075         Yes, calculat "Blocks free"
------------------------------------------------------------------
[C8A7/DB26/DD87/E433]
Correct BAM and write to diskette
EEF4   20 93 DF   JSR $DF93         Get current buffer #
EEF7   AA         TAX               Get # of corresponding
EEF8   BD 5B 02   LDA $025B,X       job codes
EEFB   29 01      AND #$01          Determine drive # and save
EEFD   85 7F      STA $7F           as current disk drive
EEFF²  A4 7F      LDY $7F           Get drive-adapted flag
EF01   B9 51 02   LDA $0251,Y       for "BAM no good"
EF04   D0 01      BNE $EF07         Must a new BAM be created?
EF06   60         RTS               No, return to main routine
EF07¹  A9 00      LDA #$00          Flag for "Invalid BAM"
EF09   99 51 02   STA $0251,Y       cleared
EF0C   20 3A EF   JSR $EF3A         Get BAM in buffer set pointer
EF0F   A5 7F      LDA $7F           Get current drive and
EF11   0A         ASL A             double that number
EF12   48         PHA               Note value
EF13   20 A5 F0   JSR $F0A5         Copy temp. storage in BAM
EF16   68         PLA               Get drive pointer
EF17   18         CLC               again
EF18   69 01      ADC #$01          Go to next temp. storage area
EF1A   20 A5 F0   JSR $F0A5         Transfer temp. storage into BAM
EF1D   A5 80      LDA $80           Retrieve current track
EF1F   48         PHA               number
EF20   A9 01      LDA #$01          Set number to
EF22   85 80      STA $80           Track 1
EF24¹  0A         ASL A             Save position of
EF25   0A         ASL A             track bytes 4 times(4 BAM
EF26   85 6D      STA $6D           bytes per track)
EF28   20 37 A9   JSR $A937         Check number of blocks free
```

```
EF2B  E6 80     INC $80         Set counter to next track and
EF2D  A5 80     LDA $80         get the
EF2F  CD AC 02  CMP $02AC       number of the last track+1
EF32  90 F0     BCC $EF24       Last track reached?
EF34  68        PLA             Rearrange old
EF35  85 80     STA $80         track number
EF37  4C 8D A5  JMP $A58D       Write BAM to diskette
-----------------------------------------------------------------
[A5AA/A738/D075/EF0C]
Read BAM and set buffer pointer
EF3A  20 0F F1  JSR $F10F       Get and note channel number
EF3D  AA        TAX             for "READ BAM"
EF3E  20 DF F0  JSR $F0DF       Set out appropriate buffer
EF41  A6 F9     LDX $F9         Get buffer number
EF43  BD E0 FE  LDA $FEE0,X     and determine memory address
EF46  85 6E     STA $6E         of buffer
EF48  A9 00     LDA #$00        Memory address put into
EF4A  85 6D     STA $6D         pointer $6D/$6E
EF4C  60        RTS             Return from this routine
-----------------------------------------------------------------
[C814/D33B]
Get number of  "BLOCKS FREE"
EF4D  A6 7F     LDX $7F         Current drive number
EF4F  BD FA 02  LDA $02FA,X     No. of free blocks (low-byte)
EF52  8D 72 02  STA $0272       received
EF55  BD FC 02  LDA $02FC,X     No. of free blocks (high-byte)
EF58  8D 73 02  STA $0273       received
EF5B  60        RTS             Return from this subroutine
-----------------------------------------------------------------
EF5C  20 F1 EF   JSR $EFF1
Unused program space from 1541 DOS
-----------------------------------------------------------------
[C87D/C8AD/CCF8]
Sector released
EF5F  4C 27 A7  JMP $A727       Sector in 1571 BAM released
EF62  38        SEC             Flag/" SECTOR already free"
EF63  D0 22     BNE $EF87       Is the block already released?
EF65  B1 6D     LDA ($6D),Y     No, get track bit pattern
EF67  1D E9 EF  ORA $EFE9,X     Release sector (bit=1)
EF6A  91 6D     STA ($6D),Y     and go back into BAM
EF6C  20 88 EF  JSR $EF88       Set flag for "BAM WRITE"
EF6F  A4 6F     LDY $6F         Current BAM byte pointer
EF71  18        CLC             Flag/"SECTOR tobe released"
EF72  B1 6D     LDA ($6D),Y     Increment # of free
EF74  69 01     ADC #$01        blocks in track and
EF76  91 6D     STA ($6D),Y     reset
EF78  A5 80     LDA $80         Get # of spur worked on and
```

```
EF7A   CD 85 FE   CMP $FE85         compare with directory track
EF7D   F0 3B      BEQ $EFBA         Identical?
EF7F   FE FA 02   INC $02FA,X       No, # of blocks on disk+1
EF82   D0 03      BNE $EF87         Verified overflow?
EF84   FE FC 02   INC $02FC,X       Overflow considered
EF87   60         RTS               Return to main routine
-----------------------------------------------------------------
[A85F/A895/EF6C/EF9F]
Set flag for "BAM on diskette illegal3"
EF88   A6 7F      LDX $7F           Determine current drive and
EF8A   A9 01      LDA #$01          set appropriate flag for
EF8C   9D 51 02   STA $0251,X       "BAM illegal"
EF8F   60         RTS               Return from this subroutine
-----------------------------------------------------------------
[CD13/EDE8/EE04/F19A/F1F2]
Lay out sector in BAM
EF90   20 F1 EF   JSR $EFF1         Write last BAM chage to disk
EF93   4C 74 A8   JMP $A874         Lay out sector in 1571 BAM
EF96   F0 36      BEQ $EFCE         Is sector already there?
EF98   B1 6D      LDA ($6D),Y       No,get byte/bit pattrn fr laid-
EF9A   5D E9 EF   EOR $EFE9,X       out blocks,layout sector (bit=0
EF9D   91 6D      STA ($6D),Y       Re-mark BAM byte
EF9F   20 88 EF   JSR $EF88         Set flag for "BAM WRITE"
EFA2   A4 6F      LDY $6F           Pointer to current BAM byte
EFA4   B1 6D      LDA ($6D),Y       No. of free sectors
EFA6   38         SEC               Decrement the track
EFA7   E9 01      SBC #$01          and rewrite
EFA9   91 6D      STA ($6D),Y       into BAM
EFAB   A5 80      LDA $80           Compar # of track being worked
EFAD   CD 85 FE   CMP $FE85         with # of directory track
EFB0   F0 0B      BEQ $EFBD         Identical?
EFB2¹  BD FA 02   LDA $02FA,X       No. of free blocks on disk
EFB5   D0 03      BNE $EFBA         Borrowing occurred?
EFB7   DE FC 02   DEC $02FC,X       Yes,decrement hibyte of counter
EFBA²  DE FA 02   DEC $02FA,X       No. of free blocks -1
EFBD¹  BD FC 02   LDA $02FC,X       No. of free blocks (high byte)
EFC0   D0 0C      BNE $EFCE         Less than 255?
EFC2   BD FA 02   LDA $02FA,X       Yes, # of free blocks (lo byte)
EFC5   C9 03      CMP #$03          Compare to three
EFC7   B0 05      BCS $EFCE         Less than three blocks free
EFC9   A9 72      DA #$72           Yes, error message of
EFCB   20 C7 E6   JSR $E6C7         "72 DISK FULL" given
EFCE³  60         RTS               Return from this subroutine
-----------------------------------------------------------------
```

```
[A845/A87B]
BAM buffer pointer set to bit for current sector and bit fetched
EFCF   20 11 F0   JSR $F011         Compute and save pointer to
EFD2   98         TYA               start of bit pattern
EFD3   85 6F      STA $6F           for track
EFD5   A5 81      LDA $81           Get # of sectors to be
EFD7   4A         LSR A             worked with, and divide
EFD8   4A         LSR A             by 8 (eight bits per byte)
EFD9   4A         LSR A             to get # of BAM bytes
EFDA   38         SEC               Add one to the
EFDB   65 6F      ADC $6F           pointer positon returned and
EFDD   A8         TAY               note the result
EFDE   A5 81      LDA $81           Get current sector number
EFE0   29 07      AND #$07          Calculate and save # of bits
EFE2   AA         TAX               per BAM byte
EFE3   B1 6D      LDA ($6D),Y       Get byte from BAM, isolate
EFE5   3D E9 EF   AND $EFE9,X       sector bit
EFE8   60         RTS               Return from this subroutine
--------------------------------------------------------------------
[A4FF/A56E/A859/A88F/D2BF/D2CB/EF67/EF9A/EFE5/F22F]
EFE9   01 02 04 08 10 20 40 80        Mask to isolate BAM bits
--------------------------------------------------------------------
[EF5C/EF90]
Write BAM to diskette
EFF1   A9 FF      LDA #$FF          Set value/"Illegal BAM"flag
EFF3   2C F9 02   BIT $02F9         Check flag status
EFF6   F0 0C      BEQ $F004         Equal to null?
EFF8   10 0A      BPL $F004         No, bit 7 cleared
EFFA   70 08      BVS $F004         No, bit 6 cleared
EFFC   A9 00      LDA #$00          Reset flag for"Illegal BAM,
EFFE   8D F9 02   STA $02F9         write new BAM"
F001   4C 0D A5   JMP $A50D         Rewrite BAM to diskette
F004[3] 60        RTS               Return from this subroutine
--------------------------------------------------------------------
[A764/BF3C/EEA3]
Clear BAM buffer
F005   20 25 A6   JSR $A625         Set pointer to BAM buffer
--------------------------------------------------------------------
[A5AD/A74C]
F008   A0 00      LDY #$00          Clear pointer to buffer positon
F00A   98         TYA               Buffer to be filled with 0
F00B[2] 91 6D     STA ($6D),Y       Write to buffer
F00D   C8         INY               Set pointer to next byte
F00E   D0 FB      BNE $F00B         Was that last byte in buffer?
F010   60         RTS               YES—return from this subroutine
--------------------------------------------------------------------
```

```
[A8B0/A90C/A925/EFCF/F130]
F011   A5 6F      LDA $6F        Zeropage addresses $6F/$70
F013   48         PHA            will be used for temp. storage
F014   A5 70      LDA $70        for this routine, and thus
F016   48         PHA            will receive
F017   A6 7F      LDX $7F        current drive number
F019   B5 FF      LDA $FF,X      and current drive status
F01B1  F0 05      BEQ $F022      Drive ready?
F01D   A9 74      LDA #$74       No, display:
F01F   20 48 E6   JSR $E648      "74 drive not ready"
F0221  20 0F F1   JSR $F10F      Determine buffer and channel #
F025   85 6F      STA $6F        Send channel number,
F027   8A         TXA            double, and
F028   0A         ASL A          send buffer
F029   85 70      STA $70        number
F02B   AA         TAX            Save value
F02C   A5 80      LDA $80        Test # of current track
F02E   DD 9D 02   CMP $029D,X    against temp.storage track data
F031   F0 0B      BEQ $F03E      Identical?
F033   E8         INX            No, chnge to next temp.
F034   86 70      STX $70        memory area
F036   DD 9D 02   CMP $029D,X    Compare track to ZS
F039   F0 03      BEQ $F03E      Are data received here?
F03B   20 5B F0   JSR $F05B      No, get track data in memory
F03E2  A5 70      LDA $70        Pointer to temp. memory
F040   A6 7F      LDX $7F        Current drive
F042   9D 9B 02   STA $029B,X    Save buffer pointer
F045   0A         ASL A          Multiply value by 4
F046   0A         ASL A          (4 bytes per entry)
F047   18         CLC            Turn BAM pointer
F048   69 A1      ADC #$A1       to position
F04A   85 6D      STA $6D        of temporary
F04C   A9 02      LDA #$02       memory
F04E   69 00      ADC #$00       Set high byte
F050   85 6E      STA $6E        or pointer
F052   A0 00      LDY #$00       Reset the current byte
F054   68         PLA            Reset zeropage addresses
F055   85 70      STA $70        $6F and $70 to
F057   68         PLA            the old
F058   85 6F      STA $6F        values
F05A   60         RTS            Return from this subroutine
---------------------------------------------------------------
```

```
[F03B]
Copy BAM bytes from BAM to temporary storage
F05B  A6 6F     LDX $6F        Get channel number
F05D  20 DF F0  JSR $F0DF      Read BAM from diskette
F060  A5 7F     LDA $7F        Get and note current
F062  AA        TAX            drive number
F063  0A        ASL A          Double number (2 drives)
F064  1D 9B 02  ORA $029B,X    Calculate old TS number,
F067  49 01     EOR #$01       Switch to another temp. area,
F069  29 03     AND #$03       and save
F06B  85 70     STA $70        New pointer
F06D  20 A5 F0  JSR $F0A5      Put actual TS contents into BAM
F070  A5 F9     LDA $F9        Current buffer number
F072  0A        ASL A          Double (pointer values—2 byte
F073  AA        TAX            numbers) and note
F074  A5 80     LDA $80        Multiply current track
F076  0A        ASL A          by four
F077  0A        ASL A          (4 BAM bytes per track)
F078  95 99     STA $99,X      Write value to buffer
F07A  A5 70     LDA $70        Current TS pointer multi-
F07C  0A        ASL A          plied by 4
F07D  0A        ASL A          (4 different TS)
F07E  A8        TAY            and set
F07F¹ A1 99     LDA ($99,X)    Get byte from BAM
F081  99 A1 02  STA $02A1,Y    and write to temp storage
F084  A9 00     LDA #$00       Clear value
F086  81 99     STA ($99,X)    in BAM
F088  F6 99     INC $99,X      Pointer to next byte
F08A  C8        INY            Pointer to next TS char.
F08B  98        TYA            Pointer checked against
F08C  29 03     AND #$03       value of 4
F08E  D0 EF     BNE $F07F      All bytes copied into TS?
F090  A6 70     LDX $70        Yes, get # of currentTS
F092  A5 80     LDA $80        Note # of corresponding
F094  9D 9D 02  STA $029D,X    track
F097  AD F9 02  LDA $02F9      Flag for "Illegal BAM"
F09A  D0 03     BNE $F09F      BAM alteration taken place?
F09C  4C 80 A4  JMP $A480      Yes, write BAM to diskette
F09F¹ 09 80     ORA #$80       Set flag for
F0A1  8D F9 02  STA $02F9      "IllegalBAM"
F0A4  60        RTS            Return from this subroutine
-----------------------------------------------------------
```

```
[EF13/EF1A/F06D]
Copy BAM bytes from temporary storage to BAM
F0A5  A8        TAY            Current TS number
F0A6  B9 9D 02  LDA $029D,Y    Get track # of TS
F0A9  F0 25     BEQ $F0D0      Is TS laid out?
F0AB  48        PHA            Yes, save track number
F0AC  A9 00     LDA #$00       Temporary storage
F0AE  99 9D 02  STA $029D,Y    freed up
F0B1  A5 F9     LDA $F9        Double current buffer
F0B3  0A        ASL A          Number(pointers are 2-byte
F0B4  AA        TAX            values)
F0B5  68        PLA            Get track number again
F0B6  0A        ASL A          and multiply by 4
F0B7  0A        ASL A          (4 BAM bytes per track)
F0B8  95 99     STA $99,X      Set pointer to track
F0BA  98        TYA            Get TS number and multiply
F0BB  0A        ASL A          by four
F0BC  0A        ASL A          (4 temp. storage areas)
F0BD  A8        TAY            and note
F0BE1 B9 A1 02  LDA $02A1,Y    Get TS byte from BAM
F0C1  81 99     STA ($99,X)    and write to buffer
F0C3  A9 00     LDA #$00       Clear value in
F0C5  99 A1 02  STA $02A1,Y    temporary storage
F0C8  F6 99     INC $99,X      Set pointer to next byte
F0CA  C8        INY            Choose next TS character
F0CB  98        TYA            Check if all 4 bytes
F0CC  29 03     AND #$03       have been transferred already
F0CE  D0 EE     BNE $F0BE      Still bytes to be copied
F0D01 60        RTS            No, return from this subroutine
-----------------------------------------------------------------
[C8F5/D042/EEB7/BF33]
Clear pointer to position of current track in BAM
F0D1  A5 7F     LDA $7F        Get current drive
F0D3  0A        ASL A          Double (2 possible drives)
F0D4  AA        TAX            and save
F0D5  A9 00     LDA #$00       Set track value=0 as flag for
F0D7  9D 9D 02  STA $029D,X    "BAM Pointer inactive"
F0DA  E8        INX            and then clear
F0DB  9D 9D 02  STA $029D,X    pointer
F0DE  60        RTS            Return from this subroutine
-----------------------------------------------------------------
[A4B0/C7BA/EF3E/F05D] Read BAM from diskette
F0DF  B5 A7     LDA $A7,X      Get buffer #, compare with
F0E1  C9 FF     CMP #$FF       flag value for "Buffer free"
F0E3  D0 25     BNE $F10A      Identical?
F0E5  8A        TXA            Yes, save channel
F0E6  48        PHA            number
```

```
F0E7    20 8E D2  JSR $D28E         Get buffer number
F0EA    AA        TAX               and save it
F0EB    10 05     BPL $F0F2         Is there a free buffer?
F0ED    A9 70     LDA #$70          No, display
F0EF    20 C8 C1  JSR $C1C8         "70 No channel" error message
F0F2[1] 86 F9     STX $F9           Set # of current buffer
F0F4    68        PLA               Get channel # again
F0F5    A8        TAY               and save it
F0F6    8A        TXA               get buffer # and enter flag
F0F7    09 80     ORA #$80          for "Buffer stil not active"
F0F9    99 A7 00  STA $00A7,Y       in table
F0FC    0A        ASL A             Double buffer number(pointers
F0FD    AA        TAX               are 2-byte values)
F0FE    AD 85 FE  LDA $FE85         Directory track
F101    95 06     STA $06,X         set as track for job
F103    A9 00     LDA #$00          sector #0
F105    95 07     STA $07,X         Set in for job
F107    4C 42 A5  JMP $A542         Block read
F10A[1] 29 0F     AND #$0F          Create and set
F10C    85 F9     STA $F9           buffer number
F10E    60        RTS               Return from this subroutine
-------------------------------------------------------------------
[D00E/EF3A/F022/F119]
Determine number of channels for BAM (in accumulator)
F10F    A9 06     LDA #$06          Get channel # for BAM channel
F111    A6 7F     LDX $7F           by drive 1 as current drive#
F113    D0 03     BNE $F118         Drive 0?
F115    18        CLC               Yes, set flag for drive #
F116    69 07     ADC #$07          and channel # for BAM
F118[1] 60        RTS               Return from this subroutine
-------------------------------------------------------------------
[A4AD/C7B7/C883/C8F8]
Determine number of channels for BAM (in X-register)
F119    20 0F F1  JSR $F10F         Determine # of channels
F11C    AA        TAX               and save in X-register
F11D    60        RTS               Return from this subroutinmme
-------------------------------------------------------------------
[D1A6/DD1D/E3A9/E3BC/E44E]
Look for next free block in BAM
F11E    20 3E DE  JSR $DE3E         Get current track & sector #
F121    A9 03     LDA #$03          Set BAM
F123    85 6F     STA $6F           pointer
F125    A9 01     LDA #$01          Set flag for"Illegal BAM,
F127    0D F9 02  ORA $02F9         Write new BAM to
F12A    8D F9 02  STA $02F9         diskette"
F12D[4] 4C DB A8  JMP $A8DB         Look for next free sector
-------------------------------------------------------------------
```

```
[A8E5/F138:A8FD,A902]
Look for next free sector
F130  20 11 F0  JSR $F011
F133  68        PLA              Get pointer again and
F134  85 6F     STA $6F          set it
F136  B1 6D     LDA ($6D),Y      # of sectors free on a track
-----------------------------------------------------------------
[A8FD/A902]
F138² D0 39     BNE $F173        Still a free sector?
F13A  A5 80     LDA $80          NO—get current track # & com-
F13C  CD 85 FE  CMP $FE85        pare with directory track #(18)
F13F  F0 19     BEQ $F15A        Identical?
F141  90 1C     BCC $F15F        No, current track # < 18?
F143  E6 80     INC $80          No, increment track # (diskette
F145  A5 80     LDA $80          built in and around 18)
F147  CD AC 02  CMP $02AC        & compare with max.# of tracks
F14A  D0 E1     BNE $F12D        Highest track # reached?
F14C  AE 85 FE  LDX $FE85        Yes, go back, label directory
F14F  CA        DEX              track -1 as
F150  86 80     STX $80          current track #
F152  A9 00     LDA #$00         Clear sector
F154  85 81     STA $81          counter
F156  C6 6F     DEC $6F          Number of blocks free
F158  D0 D3     BNE $F12D        Still a free sector?
F15A² A9 72     LDA #$72         No, display"72 DISK
F15C  20 C8 C1  JSR $C1C8        FULL" error message
F15F¹ C6 80     DEC $80          Track-by-track to outmost track
F161  D0 CA     BNE $F12D        Outermost track reached(0)?
F163  AE 85 FE  LDX $FE85        Yes, get directory track # and
F166  E8        INX              give one track more as
F167  86 80     STX $80          current track #
F169  A9 00     LDA #$00         Clear sector
F16B  85 81     STA $81          pointer (0)
F16D  C6 6F     DEC $6F          Number of free sectors
F16F  D0 BC     BNE $F12D        Still a free sector
F171  F0 E7     BEQ $F15A        No, display "Disk full"
-----------------------------------------------------------------
[F138]
Look for next free sector on this track
F173  A5 81     LDA $81          Number of current sectors
F175  18        CLC              Adopt optimal sector set-up for
F176  65 69     ADC $69          two sectors and
F178  85 81     STA $81          save as current sector number
F17A  A5 80     LDA $80          Number of current track
F17C  20 4B F2  JSR $F24B        Number of sectors comprising
F17F  8D 4E 02  STA $024E        a track determined
F182  8D 4D 02  STA $024D        and noted
```

```
F185   C5 81      CMP $81            Compare with new sector #
F187   B0 0C      BCS $F195          Number too high?
F189   38         SEC                Yes, get the #
F18A   A5 81      LDA $81            of the current sector
F18C   ED 4E 02   SBC $024E          & max. sector # transfer
F18F   85 81      STA $81            Note result as new sector #
F191   F0 02      BEQ $F195          Has sector 0 been chosen?
F193   C6 81      DEC $81            No,-correct sector variations
F195²  20 FA F1   JSR $F1FA          Look for next free sector
F198   F0 03      BEQ $F19D          Got it?
F19A¹  4C 90 EF   JMP $EF90          Yes, put sector in BAM
F19D¹  A9 00      LDA #$00           Sector # 0
F19F   85 81      STA $81            set
F1A1   20 FA F1   JSR $F1FA          Look for next free sector
F1A4   D0 F4      BNE $F19A          Found it?
F1A6   4C F5 F1   JMP $F1F5          No, display"71 Directory error"
--------------------------------------------------------------------
[DCDA]
Lay out next optimum sector
F1A9   A9 01      LDA #$01           Set flag for
F1AB   0D F9 02   ORA $02F9          "Illegal BAM" (written on
F1AE   8D F9 02   STA $02F9          diskette)
F1B1   A5 86      LDA $86            Zeropage addresses tobe used by
F1B3   48         PHA                routine & consequently reserved
F1B4   A9 01      LDA #$01           Initialize track
F1B6   85 86      STA $86            number pointer
F1B8¹  AD 85 FE   LDA $FE85          Get directory track #
F1BB   38         SEC                Draw counter / current track to
F1BC   E5 86      SBC $86            get track # above or below
F1BE   85 80      STA $80            track 18
F1C0   90 09      BCC $F1CB          Is track # les than 18 ?
F1C2   F0 07      BEQ $F1CB          No, equal to 18 ?
F1C4   4C 05 A9   JMP $A905          No, BAM pointer to sector bit
--------------------------------------------------------------------
[A90F]
F1C7   B1 6D      LDA ($6D),Y        Get # of free blocks on track
--------------------------------------------------------------------
[A91B]
F1C9¹  D0 1B      BNE $F1E6          Still some free sectors?
F1CB²  AD 85 FE   LDA $FE85          No, get #of directory track
F1CE   18         CLC                & incrment track counter, so to
F1CF   65 86      ADC $86            receive a current track #
F1D1   85 80      STA $80            above the directory track
F1D3   E6 86      INC $86            Counter for track #(next track)
F1D5   CD AC 02   CMP $02AC          compared with highest track #
```

```
F1D8   90 05      BCC $F1DF         Max track # reached?
F1DA   A9 67      LDA #$67          Yes, display "67 Illegal track
F1DC   20 45 E6   JSR $E645         or sector" error message
F1DF1  4C 1E A9   JMP $A91E         Look for next free 1571 sector
-------------------------------------------------------------------
[A928]
F1E2   B1 6D      LDA ($6D),Y       Get # of free blocks in track
-------------------------------------------------------------------
[A934]
F1E41  F0 D2      BEQ $F1B8         Still a sector free?
F1E61  68         PLA               Yes, zeropage address $86
F1E7   85 86      STA $86           rearranged
F1E9   A9 00      LDA #$00          Clear sector
F1EB   85 81      STA $81           counter
F1ED   20 FA F1   JSR $F1FA         and look for next free sector
F1F0   F0 03      BEQ $F1F5         Found?
F1F2   4C 90 EF   JMP $EF90         Yes, place sector in BAM & jump
F1F5*  A9 71      LDA #$71          diplay "71 DIR
F1F7   20 45 E6   JSR $E645         error" message
-------------------------------------------------------------------
[CD09/CD27/F195/F1A1/F1ED]
F1FA   4C A9 A8   JMP $A8A9         Look for next free track sector
-------------------------------------------------------------------
[A8B3]
F1FD1  98         TYA               Note pntr position bit patterns
F1FE   48         PHA               of blocks used
F1FF   20 20 F2   JSR $F220         Test # of blocks free
F202   A5 80      LDA $80           Number of current track
F204   20 4B F2   JSR $F220         Get # of sectors in
F207   8D 4E 02   STA $024E         this track
F20A   68         PLA               Get bit pattern pointer in
F20B   85 6F      STA $6F           BAM again
F20D1  A5 81      LDA $81           Compare # of current sector
F20F   CD 4E 02   CMP $024E         with total # of sectors
F212   B0 09      BCS $F21D         Sector # smaller?
F214   20 D5 EF   JSR $EFD5         Yes,getbit for sector f/BAM
F217   D0 06      BNE $F21F         Is the sector free?
F219   E6 81      INC $81           No,set pnter to next SECTOR
F21B   D0 F0      BNE $F20D         Jump back to $F20D
F21D1  A9 00      LDA #$00          Flag"No track sectors free"
F21F1  60         RTS               Return to main routine
-------------------------------------------------------------------
```

```
[A93E/F1FF]
Check number of free blocks in BAM for every track
F220  A5 6F     LDA $6F        Zeropage address $6F USED
F222  48        PHA            as temp. storage
F223  A9 00     LDA #$00       Clear free-blocks
F225  85 6F     STA $6F        Counter
F227  AC 86 FE  LDY $FE86      Get #of BAM bytes per track
F22A  88        DEY            & design #bytes per bit pattern
F22B1 A2 07     LDX #$07       Counter/# of bits per byte
F22D2 B1 6D     LDA ($6D),Y    Get byte from BAM & isolate
F22F  3D E9 EF  AND $EFE9,X    Bit to which bit countr pts
F232  F0 02     BEQ $F236      Is the block laid out?
F234  E6 6F     INC $6F        No,increment Free-block counter
F2361 CA        DEX            & go to the next bit
F237  10 F4     BPL $F22D      All chosen bits tested?
F239  88        DEY            Yes,set ptr tonext BAMbyte
F23A  D0 EF     BNE $F22B      All BAM bytes on trak tested?
F23C  B1 6D     LDA ($6D),Y    Yes, compare #of blocks stated
F23E  C5 6F     CMP $6F        in BAM with resulting #
F240  D0 04     BNE $F246      Identical?
F242  68        PLA            Yes, rearrange zeropage
F243  85 6F     STA $6F        Address $6F
F245  60        RTS            Return to main routine
F2461 A9 71     LDA #$71       Display
F248  20 45 E6  JSR $E645      "71 DIR error" message
----------------------------------------------------------------
[D540/D568/EED2/F17C/F204]
Get number of sectors per track
(Track number must be put into accumulator)
F24B  20 4F A7  JSR $A74F      Get # of track zones
F24E1 DD D6 FE  CMP $FED6,X    Compare max # tracks/zone with
F251  CA        DEX            actual #of tracks;change zone
F252  B0 FA     BCS $F24E      Is track larger than max zone?
F254  BD D1 FE  LDA $FED1,X    Yes,#of sectors in trackzone
F257  60        RTS            Return to main routine
----------------------------------------------------------------
[CB12/CDA3/E7A8]
F258  60        RTS            No function
----------------------------------------------------------------
[BF6C]
Execute disk controller reset
F259  A9 6F     LDA #$6F       "Sync" &"Write-protect" switched
F25B  8D 02 1C  STA $1C02      as input lines, and their
F25E  29 F0     AND #$F0       values placed in
F260  4C F8 A9  JMP $A9F8      patch
```

```
F263¹ AD 0C 1C  LDA $1C0C     Set peripheral control register
F266  29 FE     AND #$FE      CA1 "Byte ready" to neg. flank
F268  09 0E     ORA #$0E      CA2 "SOE" to high input
F26A  09 E0     ORA #$E0      CB2 (Head) set to read
F26C  8D 0C 1C  STA $1C0C     Register activated
F26F  A9 41     LDA #$41      PB7(Sync)set to output& active
F271  8D 0B 1C  STA $1C0B     Input tempstorage forHEAD data
F274  A9 00     LDA #$00      Set cnter for interrupt timer1
F276  8D 06 1C  STA $1C06     so the disk controller
F279  A9 20     LDA #$20      routine wilbe called for 8 MS,
F27B  8D 07 1C  STA $1C07     & timer 1
F27E  8D 05 1C  STA $1C05     will start
F281  A9 7F     LDA #$7F      Clear
F283  8D 0E 1C  STA $1C0E     Interrupt flag
F286  A9 C0     LDA #$C0      Interrrupt that allows
F288  8D 0D 1C  STA $1C0D     Enable "Timer 1 has
F28B  8D 0E 1C  STA $1C0E      run to zero"
F28E  A9 FF     LDA #$FF      Clear flags:
F290  85 3E     STA $3E       flag for active drive
F292  85 51     STA $51       Flag for"Format procedure" on
F294  A9 08     LDA #$08      Set identifier
F296  85 39     STA $39       Block header
F298  A9 07     LDA #$07      Set identifier for
F29A  85 47     STA $47       data block header
F29C  A9 05     LDA #$05      Call in $FA02
F29E  85 62     STA $62       Turn to
F2A0  A9 FA     LDA #$FA      routine
F2A2  85 63     STA $63       in $FA05
F2A4  A9 C8     LDA #$C8      Determine # of steps for
F2A6  85 64     STA $64       fast head movement
F2A8  A9 04     LDA #$04      Determine # of steps
F2AA  85 5E     STA $5E       to move & stop
F2AC  A9 04     LDA #$04      disk head
F2AE  85 5F     STA $5F       movement
----------------------------------------------------------------
[9DCA/9DD5/BF06]
Jump to disk controller routine
F2B0  BA        TSX           Reserve
F2B1  86 49     STX $49       stack pointer
F2B3  AD 04 1C  LDA $1C04     Reset timer
F2B6  AD 0C 1C  LDA $1C0C     CA2 (SOE=Serial Output Enable)
F2B9  09 0E     ORA #$0E      set to high
F2BB  8D 0C 1C  STA $1C0C     output
F2BE  A0 05     LDY #$05      Number of buffer
F2C0¹ B9 00 00  LDA $0000,Y   Get flag for jobcode
F2C3  10 2E     BPL $F2F3     Is there a command for buffer?
F2C5  C9 D0     CMP #$D0      Yes,compare with"PROGRAMSTART"
```

```
F2C7   D0 04      BNE $F2CD       Program executed in buffer?
F2C9   98         TYA             Yes, get buffer # &
F2CA   4C 70 F3   JMP $F370       jump to program
F2CD1  29 01      AND #$01        Get drive # from jobcode
F2CF   F0 07      BEQ $F2D8       Is the job for drive0 ?
F2D1   84 3F      STY $3F         No, note appropriate buffer
F2D3   A9 0F      LDA #$0F        Display "74 Drive not ready"
F2D5   4C 69 F9   JMP $F969       error message
F2D81  AA         TAX             Note drive
F2D9   85 3D      STA $3D         number (0) & get
F2DB   C5 3E      CMP $3E         "Drive active" flag
F2DD   F0 0A      BEQ $F2E9       Drive already running?
F2DF   20 7E F9   JSR $F97E       No, motor on
F2E2   A5 3D      LDA $3D         & set "Drive active"
F2E4   85 3E      STA $3E         flag
F2E6   4C 9C F9   JMP $F99C       Wait until motor runs
F2E91  A5 20      LDA $20         Get drive status
F2EB   30 03      BMI $F2F0       Is motor at rotation speed?
F2ED   0A         ASL A           Yes,steppermotor flagbit/carry
F2EE   10 09      BPL $F2F9       Is the head moving?
F2F01  4C 9C F9   JMP $F99C       Yes, move head into position
F2F31  88         DEY             Mark buffer #
F2F4   10 CA      BPL $F2C0       All buffer checked out?
F2F6   4C 9C F9   JMP $F99C       Yes, goto main control routine
F2F91  A9 20      LDA #$20        Flag for "Motor on"
F2FB   85 20      STA $20         set as drive status
F2FD   A0 05      LDY #$05        Determine max # of buffers
F2FF   84 3F      STY $3F         as actual buffer #
F3011  20 93 F3   JSR $F393       Set pointer to buffer address
F304   30 1A      BMI $F320       Job assignment laid out?
F3062  C6 3F      DEC $3F         No,buffer cnter to next buffer
F308   10 F7      BPL $F301       Last buffer reached?
F30A   A4 41      LDY $41         YES—set last job #
F30C   20 95 F3   JSR $F395       at buffer pointer
F30F   A5 42      LDA $42         Get track diff. to last job
F311   85 4A      STA $4A         & set # to stepper half-steps
F313   06 4A      ASL $4A         to be executed
F315   A9 60      LDA #$60        Flag for head movement step
F317   85 20      STA $20         in drive status
F319   B1 32      LDA ($32),Y     Get & mark track # for job
F31B   85 22      STA $22         from buffer
F31D   4C 9C F9   JMP $F99C       Position Head on track
F3201  29 01      AND #$01        Design drive number
F322   C5 3D      CMP $3D         & compare to last job drive
F324   D0 E0      BNE $F306       Is the job for the same drive?
F326   A5 22      LDA $22         Get track # of last job
F328   F0 12      BEQ $F33C       Track # on hand?
```

```
F32A   38         SEC                   Yes,figure difference between
F32B   F1 32      SBC ($32),Y           current & last track
F32D   F0 0D      BEQ $F33C             Is the job for same track?
F32F   49 FF      EOR #$FF              Produce # / stepper increments
F331   85 42      STA $42               & store
F333   E6 42      INC $42               number
F335   A5 3F      LDA $3F               Transfer drive # of current
F337   85 41      STA $41               job
F339   4C 06 F3   JMP $F306             Work with next buffer
F33C2  A2 04      LDX #$04              Number of different trackzones
F33E   B1 32      LDA ($32),Y           Get track # of job
F340   85 40      STA $40               & mark it
F3421  DD D6 FE   CMP $FED6,X           Compare with highest zone track
F345   CA         DEX                   Set zone counter to next zone
F346   B0 FA      BCS $F342             Track lie within zone?
F348   BD D1 FE   LDA $FED1,X           Yes,get # of sectors per zone
F34B   85 43      STA $43               & set
F34D   8A         TXA                   Zone numbers (0-3)
F34E   0A         ASL A                 will be in
F34F   0A         ASL A                 bits 5 & 6
F350   0A         ASL A                 The bit exchange rate,
F351   0A         ASL A                 which (with  help of head
F352   0A         ASL A                 electronics)writes to disk
F353   85 44      STA $44               dictates value in temp memory
F355   AD 00 1C   LDA $1C00             Get drive control register
F358   29 9F      AND #$9F              Clear bits for bitrate & set
F35A   05 44      ORA $44               zone chosen with the value
F35C   8D 00 1C   STA $1C00             used
F35F   A6 3D      LDX $3D               Drive #
F361   A5 45      LDA $45               Get jobcode
F363   C9 40      CMP #$40              Compare with "Head to track 1"
F365   F0 15      BEQ $F37C             Should the head be reset?
F367   C9 60      CMP #$60              No, code for external job prg.
F369   F0 03      BEQ $F36E             Program taken into buffer?
F36B   4C B1 F3   JMP $F3B1             No, return to main routine
-----------------------------------------------------------------
[F369]  VGL. 93A2
Start program in buffer
F36E   A5 3F      LDA $3F               Get current buffer #
-----------------------------------------------------------------
```

```
[F2CA]
Start program (Buffer address in A)
F370  18        CLC              & compute the high
F371  69 03     ADC #$03         byte of the absolute
F373  85 31     STA $31          buffer address
F375  A9 00     LDA #$00         Set low byte
F377  85 30     STA $30          to null
F379  6C 30 00  JMP ($0030)      Jump to buffer program
------------------------------------------------------------------
[F365]  VGL. 93B0
Move head back to track 1
F37C  A9 60     LDA #$60         Set head flag for movement
F37E  85 20     STA $20          in drive status
F380  AD 00 1C  LDA $1C00        Get drive dontrol register
F383  29 FC     AND #$FC         & clear stepper
F385  8D 00 1C  STA $1C00        impulse
F388  A9 A4     LDA #$A4         Number of steps (92)
F38A  85 4A     STA $4A          Set to outside(= 46 tracks)
F38C  A9 01     LDA #$01         Set track 1
F38E  85 22     STA $22          as job track #
F390  4C 69 F9  JMP $F969        Close job loop
------------------------------------------------------------------
[BF0C/F301/F43A/F48F]  VGL. 93D1
Initialize buffer pointer for job
F393  A4 3F     LDY $3F          Number of current buffer
------------------------------------------------------------------
[F30C]
Set buffer pointer (Buffer number in Y)
F395  B9 00 00  LDA $0000,Y      Get appropriate jocode
F398  48        PHA              & note
F399  10 10     BPL $F3AB        Job on hand?
F39B  29 78     AND #$78         Yes, isolate and mark command
F39D  85 45     STA $45          Bits for disk controller
F39F  98        TYA              Get buffer #
F3A0  0A        ASL A            & double (2-byte value)
F3A1  69 06     ADC #$06         Compute track & sector table
F3A3  85 32     STA $32          & set pointers
F3A5  98        TYA              Get buffer # again, &
F3A6  18        CLC              calculate the physical
F3A7  69 03     ADC #$03         memory address of the
F3A9  85 31     STA $31          buffer (high-byte) & set
F3AB1 A0 00     LDY #$00         low byte of the pointer
F3AD  84 30     STY $30          to null
F3AF  68        PLA              Get jobcode
F3B0  60        RTS              Return to main routine
------------------------------------------------------------------
```

```
[F36B/F5E6]
Look for track.
(Routine will place information on every block header on diskette).
F3B1    A2 5A     LDX #$5A      No. of read searches(90)
F3B3    86 4B     STX $4B       determined
F3B5    A2 00     LDX #$00      Clear # of header bytes
F3B7    A9 52     LDA #$52      Save GCR identifier for
F3B9    85 24     STA $24       block header
F3BB[1] 20 56 F5  JSR $F556     Wait for synch-marker
F3BE[1] 50 FE     BVC $F3BE     Read electronics ready?
F3C0    B8        CLV           Yes, set flag back
F3C1    AD 01 1C  LDA $1C01     Read header from disk—compare
F3C4    C5 24     CMP $24       with block identifier
F3C6    D0 3F     BNE $F407     Is ther a blockheader?
F3C8[2] 50 FE     BVC $F3C8     Yes, wait for next byte
F3CA    B8        CLV           Reactivate reading electronics
F3CB    AD 01 1C  LDA $1C01     Read byte from diskette
F3CE    95 25     STA $25,X     & store in header buffer
F3D0    E8        INX           Increment counter
F3D1    E0 07     CPX #$07      Compare with # of headerbytes
F3D3    D0 F3     BNE $F3C8     Entire header read?
F3D5    20 97 F4  JSR $F497     Yes—convert header fr GCR to %
F3D8    A0 04     LDY #$04      Set pnter to checksum position
F3DA    A9 00     LDA #$00      Compute header
F3DC[1] 59 16 00  EOR $0016,Y   checksum
F3DF    88        DEY           Pointer to next header byte
F3E0    10 FA     BPL $F3DC     All bytes computed?
F3E2    C9 00     CMP #$00      YES—value for erroe-free header
F3E4    D0 38     BNE $F41E     Checksum error occurred?
F3E6    A6 3E     LDX $3E       NO—get current drive number
F3E8    A5 18     LDA $18       TRack number of header to be read
F3EA    95 22     STA $22,X     saved as current track
F3EC    A5 45     LDA $45       get jobcode
F3EE    C9 30     CMP #$30      Conmpare with"Read Sector"
F3F0    F0 1E     BEQ $F410     Identical?
F3F2    A5 3E     LDA $3E       NO—get drive number of job
F3F4    0A        ASL A         Turn pointer to drive with
F3F5    A8        TAY           corresponding ID
F3F6    B9 12 00  LDA $0012,Y   Get 1st char of ID and compare
F3F9    C5 16     CMP $16       with blockheader ID
F3FB    D0 1E     BNE $F41B     ID been changed?
F3FD    B9 13 00  LDA $0013,Y   NO—get next ID char and compare
F400    C5 17     CMP $17       with header ID
F402    D0 17     BNE $F41B     Identical?
F404    4C 23 F4  JMP $F423     YES—determine next job
F407[1] C6 4B     DEC $4B       Decrement read-search counter
F409    D0 B0     BNE $F3BB     90 read searches executed
```

```
F40B  A9 02     LDA #$02          Display error message
F40D  20 69 F9  JSR $F969         "20 Read error"
F410¹ A5 16     LDA $16           Take on blockheader ID
F412  85 12     STA $12           as new ID for
F414  A5 17     LDA $17           current
F416  85 13     STA $13           disk drive
F418² A9 01     LDA #$01          Number for "OK"
F41A  2C        .BYTE $2C         Two-byte jump (bit command)
F41B² A9 0B     LDA #$0B          # for"29 Disk ID mismatch"
F41D  2C        .BYTE $2C         Two-byte jump (bit command)
F41E¹ A9 09     LDA #$09          Number for "27 Write error"
F420  4C 69 F9  JMP $F969         message returned
--------------------------------------------------------------
[F404]  VGL. 94BC
Get next optimum job
F423  A9 7F     LDA #$7F          Intitialize pntr for difference
F425  85 4C     STA $4C           to next job
F427  A5 19     LDA $19           Get sector # from last blkheader
F429  18        CLC               and compare with
F42A  69 02     ADC #$02          maximum
F42C  C5 43     CMP $43           sector number
F42E  90 02     BCC $F432         Is number in allowed range?
F430  E5 43     SBC $43           NO—subtract max. sector number &
F432¹ 85 4D     STA $4D           save new sector number
F434  A2 05     LDX #$05          Set buffer
F436  86 3F     STX $3F           number
F438  A2 FF     LDX #$FF          buffer pointer
F43A¹ 20 93 F3  JSR $F393         Set buffer address & get jobcode
F43D  10 44     BPL $F483         Job available?
F43F  85 44     STA $44           YES—Save jobcode and determine
F441  29 01     AND #$01          Drive number of the job
F443  C5 3E     CMP $3E           Comparable with actual drive?
F445  D0 3C     BNE $F483         Is the job for current drive?
F447  A0 00     LDY #$00          YES—clear buffer pointer
F449  B1 32     LDA ($32),Y       Compare track number of the job
F44B  C5 40     CMP $40           with last track
F44D  D0 34     BNE $F483         Identical?
F44F  A5 45     LDA $45           YES—get jobcode command bits
F451  C9 60     CMP #$60          Code for "Program in buffer"
F453  F0 0C     BEQ $F461         SHould buffer program be run?
F455  A0 01     LDY #$01          NO—pointer to params for buffer 0
F457  38        SEC               Get sector number of job
F458  B1 32     LDA ($32),Y       for buffer 0 and compare
F45A  E5 4D     SBC $4D           wirth optimum sectors computed
F45C  10 03     BPL $F461         Is new sector number less?
F45E  18        CLC               NO—calculate # of sectors up to
F45F  65 43     ADC $43           this sector and compare
```

```
F461²  C5 4C     CMP $4C        with last difference
F463   B0 1E     BCS $F483      Is new value less than last?
F465   48        PHA            YES—Save sector difference
F466   A5 45     LDA $45        Check command bits of jobcode
F468   F0 14     BEQ $F47E      Should sector be read?
F46A   68        PLA            NO—get difference again and
F46B   C9 09     CMP #$09       Compare to 9
F46D   90 14     BCC $F483      Is value less?
F46F   C9 0C     CMP #$0C       NO—compare to 13
F471   B0 10     BCS $F483      Is difference <13 ?
F473¹  85 4C     STA $4C        YES—Save new sector difference
F475   A5 3F     LDA $3F        Get buffer number of the job and
F477   AA        TAX            compute
F478   69 03     ADC #$03       the physical memory
F47A   85 31     STA $31        address (high-byte)
F47C   D0 05     BNE $F483      Jump to $F483
F47E¹  68        PLA            Get sector difference and
F47F   C9 06     CMP #$06       compare to 6
F481   90 F0     BCC $F473      Is difference larger?
F483⁷  C6 3F     DEC $3F        YES—turn pointer to next pointer
F485   10 B3     BPL $F43A      All buffers tested yet?
F487   8A        TXA            YES—get buffer # of next job
F488   10 03     BPL $F48D      Optimum job found?
F48A   4C 9C F9  JMP $F99C      NO—execute stepper command
F48D¹  86 3F     STX $3F        Save buffer number
F48F   20 93 F3  JSR $F393      Compute buffer address
F492   A5 45     LDA $45        Get clear jobcode
F494   4C CA F4  JMP $F4CA      Execute read/write jobs
-----------------------------------------------------------------
[F3D5]
Convert header from GCR-code into binary values
F497   A5 30     LDA $30        Retrieve
F499   48        PHA            pointer to
F49A   A5 31     LDA $31        current
F49C   48        PHA            buffer address
F49D   A9 24     LDA #$24       Adjust pointer at
F49F   85 30     STA $30        $0024 (start of data for
F4A1   A9 00     LDA #$00       last-read
F4A3   85 31     STA $31        blockheaders)
F4A5   A9 00     LDA #$00       Reset buffer pointer for
F4A7   85 34     STA $34        conversion routine
F4A9   20 E6 F7  JSR $F7E6      Convert 5 GCR bytes to 4binary#S
F4A7   A5 55     LDA $55        4th byte converted to
F4AE   85 18     STA $18        track number in header buffer
F4B0   A5 54     LDA $54        Third byte is
F4B2   85 19     STA $19        sector number in header buffer
```

```
F4B4  A5 53     LDA $53          Second byte is
F4B6  85 1A     STA $1A          checksum in header buffer
F4B8  20 E6 F7  JSR $F7E6        Convrt 5 GCRbytes to 4 binary #'s
F4BB  A5 52     LDA $52          First byte
F4BD  85 17     STA $17          is 2nd ID char in header buffer
F4BF  A5 53     LDA $53          Second byte is
F4C1  85 16     STA $16          first ID char in header buffer
F4C3  68        PLA              Re-create pointer
F4C4  85 31     STA $31          to address of
F4C6  68        PLA              current
F4C7  85 30     STA $30          buffer
F4C9  60        RTS              Return from this subroutine
-----------------------------------------------------------------
[F494]  VGL. 9606
Read sector from diskette to buffer
F4CA  C9 00     CMP #$00         Compare jobcode with readcode
F4CC  F0 03     BEQ $F4D1        Identical?
F4CE  4C 6E F5  JMP $F56E        NO—test jobcode further
-----------------------------------------------------------------
Read sector
F4D1[1] 20 0A F5  JSR $F50A      Ssearch for sector blockheader
F4D4[2] 50 FE     BVC $F4D4      Wait for byte from disk
F4D6  B8        CLV              Read electronics ready to
F4D7  AD 01 1C  LDA $1C01        read byte with dish head
F4DA  91 30     STA ($30),Y      and write to current buffer
F4DC  C8        INY              Set buffer pointer to next byte
F4DD[1] D0 F5     BNE $F4D4      Buffer already full?
F4DF  A0 BA     LDY #$BA         YES—set buff pntr to cond'l buff
F4E1[1] 50 FE     BVC $F4E1      Wait for next byte from disk
F4E3  B8        CLV              Get flag to signal byte
F4E4  AD 01 1C  LDA $1C01        from read head
F4E7  99 00 01  STA $0100,Y      & write to conditional buffer
F4EA  C8        INY              Set buffer pointer to next byte
F4EB  D0 F4     BNE $F4E1        Conditional buffer full?
F4ED  20 E0 F8  JSR $F8E0        YES—convert sector frm GCR»binary
F4F0  A5 38     LDA $38          Get 1st byte of data block &
F4F2  C5 47     CMP $47          identifier for data blockheader
F4F4  F0 05     BEQ $F4FB        Data block ?
F4F6  A9 04     LDA #$04         NO—display error message:
F4F8  4C 69 F9  JMP $F969        "22 Read error"
F4FB[1] 20 E9 F5  JSR $F5E9      Compare checksum computed for
F4FE  C5 3A     CMP $3A          data with value read in
F500  F0 03     BEQ $F505        Identical?
F502  A9 05     LDA #$05         Error # for"23 read error"
F504  2C        .BYTE $2C        Jump to next 2 bytes(Bir Command)
F505  A9 01     LDA $01          Error number for "OK"
F507  4C 69 F9  JMP $F969        message given
```

```
[F4D1/F6A0]  CF. 9600
Set read-head into position after data block sync-marking a sector
F50A  20 10 F5  JSR $F510        Search for a sector blockheader
F50D  4C 56 F5  JMP $F556        Wait f/sync-mark of a data block
--------------------------------------------------------------------
[F50A/F589/F6CA] VGL. 970F
Look for sector header
F510  A5 3D     LDA $3D          Get drive number of job
F512  0A        ASL A            and corresponding
F513  AA        TAX              ID
F514  B5 12     LDA $12,X        First ID character
F516  85 16     STA $16          transferred to jheader buffer
F518  B5 13     LDA $13,X        Second ID character
F51A  85 17     STA $17          transferred to header buffer
F51C  A0 00     LDY #$00         Clear buffer pointer
F51E  B1 32     LDA ($32),Y      Get track # frm current buffr &
F520  85 18     STA $18          transfer to jheader buffer
F522  C8        INY              Set buffer pointer to next char
F523  B1 32     LDA ($32),Y      Get sector # from curr. buffr
F525  85 19     STA $19          & transfer to header buffer
F527  A9 00     LDA #$00         Calculate checksum of
F529  45 16     EOR $16          sector header made
F52B  45 17     EOR $17          available
F52D  45 18     EOR $18          and write
F52F  45 19     EOR $19          to
F531  85 1A     STA $1A          header buffer
F533  20 34 F9  JSR $F934        Convrt sector header to GCRbytes
F536  A2 5A     LDX #$5A         Number of read searches(90)
F538¹ 20 56 F5  JSR $F556        Wait for next sync-marking
F53B  A0 00     LDY #$00         Clear buffer pointer
F53D² 50 FE     BVC $F53D        Wait for next byte from disk
F53F  B8        CLV              Get flag again for
F540  AD 01 1C  LDA $1C01        byte from read head
F543  D9 24 00  CMP $0024,Y      & compare with available header
F546  D0 06     BNE $F54E        Values identical?
F548  C8        INY              YES—set buffer pntr to next char
F549  C0 08     CPY #$08         Compare with # of header bytes
F54B  D0 F0     BNE $F53D        Entire header tested?
F54D  60        RTS              Return from this subroutine
F54E¹ CA        DEX              Decrement read search counter
F54F  D0 E7     BNE $F538        Any more read searches?
F551  A9 02     LDA #$02         Display error message:
F553¹ 4C 69 F9  JMP $F969        "20 Read error"
--------------------------------------------------------------------
```

```
[BF1E/F3BB/F50D/F538/FB1D/FD39/FD62]  CF. 9754
Wait for next sync-mark
F556   A9 D0      LDA #$D0          Set timer to about 53 MS and
F558   8D 05 18   STA $1805         start
F55B   A9 03      LDA #$03          Number for "21 Read error"
F55D¹  2C 05 18   BIT $1805         Get condition of timer
F560   10 F1      BPL $F553         Is timer running?
F562   2C 00 1C   BIT $1C00         Get condition of sync-flag
F565   30 F6      BMI $F55D         Has sync-mark been found?
F567   AD 01 1C   LDA $1C01         YES—intialize head
F56A   B8         CLV               Read electronic readied again
F56B   A0 00      LDY #$00          Set processor flags
F56D   60         RTS               Return from this subroutine
-----------------------------------------------------------------
[F4CE]  cf. 976E
Write sector when jobcode $90 (Command bit $10)
F56E   C9 10      CMP #$10          Compare with 'write' jobcode
F570   F0 03      BEQ $F575         Identical?
F572   4C 91 F6   JMP $F691         NO—Jobcode search continues
-----------------------------------------------------------------------
Write sector
F575¹  20 E9 F5   JSR $F5E9         Compute buffer checksum
F578   85 3A      STA $3A           and save it
F57A   AD 00 1C   LDA $1C00         drive control register
F57D   29 10      AND #$10          Get 'Write Protect' bit flag
F57F   D0 05      BNE $F586         Is there a write protect?
F581   A9 08      LDA #$08          YES—Display error message:
F583   4C 69 F9   JMP $F969         '26 Write Protect On'
F586¹  20 8F F7   JSR $F78F         Convert buffer to GCR-Code
F589   20 10 F5   JSR $F510         Search block header of sector
F58C   A2 09      LDX #$09          Number of bytes on header
F58E²  50 FE      BVC $F58E         Byte read from diskette?
F590   B8         CLV               YES—Byte Ready set up
F591   CA         DEX               Read over next byte
F592   D0 FA      BNE $F58E         Entire block header jumped over?
F594   A9 FF      LDA #$FF          YES—Switch register for  head
F596   8D 03 1C   STA $1C03         to output
F599   AD 0C 1C   LDA $1C0C         Get drive control register
F59C   29 1F      AND #$1F          Place controller circuitry
F59E   09 C0      ORA #$C0          on write mode and
F5A0   8D 0C 1C   STA $1C0C         set in register
F5A3   A9 FF      LDA #$FF          Sync-marking value
F5A5   A2 05      LDX #$05          Number of sync-bytes for  marking
F5A7   8D 01 1C   STA $1C01         Transfer byte to head
F5AA   B8         CLV               Prepare  Byte Ready flag
F5AB²  50 FE      BVC $F5AB         Wait until byte is written
F5AD   B8         CLV               Prepare Byte Ready flag
```

```
F5AE   CA         DEX          Counter for number of sync-bytes
F5AF   D0 FA      BNE $F5AB    All sync-bytes on diskette?
F5B1   A0 BB      LDY #$BB     YES—Buffr pointr to status buffer
F5B3¹  B9 00 01   LDA $0100,Y  Get byte from buffer
F5B6¹  50 FE      BVC $F5B6    Wait til write circuitry is ready
F5B8   B8         CLV          Flag reset
F5B9   8D 01 1C   STA $1C01    Write byte to diskette
F5BC   C8         INY          Pointer to next char in buffer
F5BD   D0 F4      BNE $F5B3    Entire buffer written?
F5BF¹  B1 30      LDA ($30),Y  YES—Get byte from file buffer
F5C1¹  50 FE      BVC $F5C1    Wait until diskette is ready
F5C3   B8         CLV          Flag reset and
F5C4   8D 01 1C   STA $1C01    write byte to diskette
F5C7   C8         INY          Pointer to next byte in buffer
F5C8   D0 F5      BNE $F5BF    Entire buffer written up?
F5CA¹  50 FE      BVC $F5CA    YES—Wait til last byte is
F5CC   AD 0C 1C   LDA $1C0C    completely written and then
F5CF   09 E0      ORA #$E0     switch controller circuitry
F5D1   8D 0C 1C   STA $1C0C    to read
F5D4   A9 00      LDA #$00     Switch read head register
F5D6   8D 03 1C   STA $1C03    to input
F5D9   20 F2 F5   JSR $F5F2    Convert buffer from GCR to binary
F5DC   A4 3F      LDY $3F      Current buffer number
F5DE   B9 00 00   LDA $0000,Y  Get jobcode for it and
F5E1   49 30      EOR #$30     establish jobcode
F5E3   99 00 00   STA $0000,Y  for 'Verify'
F5E6   4C B1 F3   JMP $F3B1    Check execution
---------------------------------------------------------------
[96FD/9775/989E/9C1B/BF2A/F4FB/F575/F698/FCA2]
Calculate buffer checksum
F5E9   A9 00      LDA #$00     Clear checksum value and
F5EB   A8         TAY          pointer to buffer position
F5EC¹  51 30      EOR ($30),Y  Compute byte from buffer checksum
F5EE   C8         INY          Set pointer to next byte
F5EF   D0 FB      BNE $F5EC    Entire buffer calculated?
F5F1   60         RTS          YES—Return from subroutine
---------------------------------------------------------------
[F5D9/F972]  vgl. 97F9
Data buffer and status buffer converted from GCR to binary
F5F2   A9 00      LDA #$00     Initialize low-byte of pointer
F5F4   85 2E      STA $2E      forthe current data buffer and
F5F6   85 30      STA $30      status buffer
F5F8   85 4F      STA $4F      Retain momentary value of pointer
F5FA   A5 31      LDA $31      to current data buffer
F5FC   85 4E      STA $4E      in $4E/$4F
F5FE   A9 01      LDA #$01     Set buffer pointer
F600   85 31      STA $31      of $1BB
```

```
F602   85 2F      STA $2F        High-byte of status buffer
F604   A9 BB      LDA #$BB       Turn buffr pointer for conversion
F606   85 34      STA $34        to start of status buffer
F608   85 36      STA $36        Set pntr to curr binary byte pos.
F60A   20 E6 F7   JSR $F7E6      Convert 5 GCRbytes to 4binary #'s
F60D   A5 52      LDA $52        Get 1st converted byte & save as
F60F   85 38      STA $38        identifier for data blockheader
F611   A4 36      LDY $36        Get buffer pointer
F613   A5 53      LDA $53        Get 2nd byte to be converted and
F615   91 2E      STA ($2E),Y    write to temporary buffer
F617   C8         INY            Set buffer pointer to next byte
F618   A5 54      LDA $54        Get 3rd converted byte
F61A   91 2E      STA ($2E),Y    and write to temporary buffer
F61C   C8         INY            Pointer to next byte
F61D   A5 55      LDA $55        Get last converted byte and
F61F   91 2E      STA ($2E),Y    store in temporary buffer
F621   C8         INY            Pointr to next position in buffer
F622   84 36      STY $36        --mark it
F624[1] 20 E6 F7  JSR $F7E6      Convert next 5 GCR-bytes
F627   A4 36      LDY $36        Get buffer pointer
F629   A5 52      LDA $52        Get 1st converted byte and
F62B   91 2E      STA ($2E),Y    write to temporary buffer
F62D   C8         INY            Set pointer to next byte
F62E   A5 53      LDA $53        Get 2nd converted byte
F630   91 2E      STA ($2E),Y    and write to temporary buffer
F632   C8         INY            Set pointer to next byte
F633   F0 0E      BEQ $F643      All temp. buffer bytes gotten?
F635   A5 54      LDA $54        NO—Get 3rd converted byte and
F637   91 2E      STA ($2E),Y    write to temp. buffer
F639   C8         INY            Buffer pointer on next byte pos.
F63A   A5 55      LDA $55        Get 4th converted byte and
F63C   91 2E      STA ($2E),Y    write to temp. buffer
F63E   C8         INY            Pointer to next byte in buffer
F63F   84 36      STY $36        and save it
F641   D0 E1      BNE $F624      Last byte from temporary buffer?
F643[1] A5 54     LDA $54        YES—Get 3rd converted byte
F645   91 30      STA ($30),Y    and write to data buffer
F647   C8         INY            Set buffer pointer to next byte
F648   A5 55      LDA $55        Get last converted byte and
F64A   91 30      STA ($30),Y    write to data buffer
F64C   C8         INY            Set buffer pointer to next char
F64D   84 36      STY $36        and save it
F64F[1] 20 E6 F7  JSR $F7E6      Next 5 GCR-bytes into binary
F652   A4 36      LDY $36        Get buffer pointer
F654   A5 52      LDA $52        Get 1st converted byte and
F656   91 30      STA ($30),Y    write to data buffer
F658   C8         INY            Set buffer pointer to next byte
```

```
F659  A5 53     LDA $53       Get 2nd converted byte
F65B  91 30     STA ($30),Y   Write to data buffer
F65D  C8        INY           Correct buffer pointer
F65E  A5 54     LDA $54       Get 3rd converted byte
F660  91 30     STA ($30),Y   Write in data buffer
F662  C8        INY           Pointer to next byte in buffer
F663  A5 55     LDA $55       Get last converted byte
F665  91 30     STA ($30),Y   Write in data buffer
F667  C8        INY           Set buffer pointer to next byte
F668  84 36     STY $36       and save it
F66A  C0 BB     CPY #$BB      Compar  buffr pointer w/end value
F66C  90 E1     BCC $F64F     All bytes converted into binary?
F66E  A9 45     LDA #$45      YES—Pointer set to
F670  85 2E     STA $2E       destination address
F672  A5 31     LDA $31       of shift operations
F674  85 2F     STA $2F       to follow
F676  A0 BA     LDY #$BA      Buffr pointr to begin/data buffer
F678¹ B1 30     LDA ($30),Y   Get byte frm lowst part of buffer
F67A  91 2E     STA ($2E),Y   shift to uppermost part
F67C  88        DEY           Pointer to next character
F67D  D0 F9     BNE $F678     Entire lower section copied?
F67F  B1 30     LDA ($30),Y   Copy lowest byte
F681  91 2E     STA ($2E),Y   into highest part
F683¹ A2 BB     LDX #$BB      Set buffr pointer to status buffr
F685¹ BD 00 01  LDA $0100,X   Get byte frm status buffer and
F688  91 30     STA ($30),Y   put in lowest free data buffer
F68A  C8        INY           Increment status buffer pointer &
F68B  E8        INX           increment data buffer pointeer
F68C  D0 F7     BNE $F685     Entire stats buffr in data buffr?
F68E  86 50     STX $50       YES—Clr'Buffer in GCR-Code' flag
F690  60        RTS           Return from subroutine
-----------------------------------------------------------------
[F572]  cf. 9898
Compare sector from diskette w/ buffer contents, when jobcode $A0
F691  C9 20     CMP #$20      Compare jobcode w/ 'Verify' code
F693  F0 03     BEQ $F698     Identical?
F695  4C CA F6  JMP $F6CA     NO—Decode jobcode further
-----------------------------------------------------------------
Sector verify
F698¹ 20 E9 F5  JSR $F5E9     Compute data buffer checksum
F69B  85 3A     STA $3A       and save it
F69D  20 8F F7  JSR $F78F     Convert buffer to GCR-code
F6A0  20 0A F5  JSR $F50A     Set head to sector start on disk
F6A3  A0 BB     LDY #$BB      Buffr pointr to start/stats buffr
F6A5¹ B9 00 01  LDA $0100,Y   Get byte from status buffer
F6A8¹ 50 FE     BVC $F6A8     Wait til byte from disk is ready
F6AA  B8        CLV           Get flag ready again
```

```
F6AB   4D 01 1C   EOR $1C01       Get byte from head and compare
F6AE   D0 15      BNE $F6C5       Byte from buffer & disk equal?
F6B0   C8         INY             YES—Pointer to next buffer byte
F6B1   D0 F2      BNE $F6A5       Entire status buffer compared?
F6B3¹  B1 30      LDA ($30),Y     YES—Get byte from data  buffer
F6B5¹  50 FE      BVC $F6B5       Wait until byte is read from disk
F6B7   B8         CLV             and get head ready again
F6B8   4D 01 1C   EOR $1C01       Get byte from head abd compare
F6BB   D0 08      BNE $F6C5       Byte from disk and buffer equal?
F6BD   C8         INY             YES—Buffer pointer to next char
F6BE   C0 FD      CPY #$FD        Compare with end value of buffer
F6C0   D0 F1      BNE $F6B3       All bytes compared?
F6C2   4C 18 F4   JMP $F418       Verify if successful
F6C5¹  A9 07      LDA #$07        Display error message --
F6C7   4C 69 F9   JMP $F969       '25 Write Error'
-------------------------------------------------------------------
[F695]  cf. 98CE
Look for sector header (jobcode $B0)
F6CA   20 10 F5   JSR $F510       Look for sector header
F6CD   4C 18 F4   JMP $F418       Prepare return message
-------------------------------------------------------------------
[F7E3/FE64/F7BC/F950/F961/FE5E]
Convert 4 Binary bytes into 5 GCR-bytes.  $52-$55 will be used
as buffer for the binary values
F6D0   A9 00      LDA #$00        Clear temporary
F6D2   85 57      STA $57         memory storage for
F6D4   85 5A      STA $5A         GCR-bytes
F6D6   A4 34      LDY $34         Pointer to current GCR-byte
F6D8   A5 52      LDA $52         Get first character
F6DA   29 F0      AND #$F0        to be converted from
F6DC   4A         LSR A           binary buffer;
F6DD   4A         LSR A           isolate most significant part of
F6DE   4A         LSR A           byte (bits 4-7) and copy to least
F6DF   4A         LSR A           significant part
F6E0   AA         TAX             then get the halfbytes of the
F6E1   BD 7F F7   LDA $F77F,X     corresponding 5-bit-GCR-code
F6E4   0A         ASL A           Copy the 5 bits into the
F6E5   0A         ASL A           higher part
F6E6   0A         ASL A           and save parts of
F6E7   85 56      STA $56         bytes (bits 3-7)
F6E9   A5 52      LDA $52         Get first byte to be converted &
F6EB   29 0F      AND #$0F        isolate lowest part;
F6ED   AA         TAX             Then pass it to half-byte
F6EE   BD 7F F7   LDA $F77F,X     Get 5-bit-GCR code
```

```
F6F1  6A        ROR A         Two lowest bits,when there is no
F6F2  66 57     ROR $57       more room if 1st byte(8 bits turn
F6F4  6A        ROR A         to 10 bits)-bring them into the
F6F5  66 57     ROR $57       second GCR-byte
F6F7  29 07     AND #$07      Combine the 3 remaining bits into
F6F9  05 56     ORA $56       the first GCR-byte
F6FB  91 30     STA ($30),Y   and write GCR-byte to buffer
F6FD  C8        INY           Buffer pointer to next character
F6FE  A5 53     LDA $53       Get second byte for conversion
F700  29 F0     AND #$F0      Get 1st part to be converted
F702  4A        LSR A         & move to least significant
F703  4A        LSR A         half-byte
F704  4A        LSR A         Equivalent binary byte to
F705  4A        LSR A         be used for pointer
F706  AA        TAX           Get corresponding
F707  BD 7F F7  LDA $F77F,X   5-bit-GCR-code and
F70A  0A        ASL A         Set in 2nd GCR-byte
F70B  05 57     ORA $57       in bit positions 1-5
F70D  85 57     STA $57       --
F70F  A5 53     LDA $53       Get 3rd byte to be converted and
F711  29 0F     AND #$0F      isolate least significant part,
F713  AA        TAX           then get corresponding
F714  BD 7F F7  LDA $F77F,X      5-bit GCR-byte
F717  2A        ROL A            Set GCR-byte
F718  2A        ROL A         in bit positions 4-7
F719  2A        ROL A         of the 3rd GCR-byte
F71A  2A        ROL A         and
F71B  85 58     STA $58       save
F71D  2A        ROL A         Transfer last GCR-bit
F71E  29 01     AND #$01      to next
F720  05 57     ORA $57       GCR-byte
F722  91 30     STA ($30),Y   Write GCR-byte to buffer
F724  C8        INY           Set buffer pointer to next byte
F725  A5 54     LDA $54       Get 3rd bin. byte to be converted
F727  29 F0     AND #$F0      and isolate most significant
F729  4A        LSR A         parts (bits 4-7)
F72A  4A        LSR A         Shift half-byte(least sig.) and
F72B  4A        LSR A         Set up pointer for equivalent
F72C  4A        LSR A         binary bytes, and
F72D  AA        TAX           half-byte as corresponding
F72E  BD 7F F7  LDA $F77F,X   5-bit GCR code
F731  18        CLC           byte shifted 1 place to the right,
F732  6A        ROR A         and a null bit inserted
F733  05 58     ORA $58       GCR-value w/previous combinations
F735  91 30     STA ($30),Y   Write GCR-byte to buffer and
F737  C8        INY           increment buffer pointer
F738  6A        ROR A         Get previously-moved bit 0 and
```

```
F739  29 80     AND #$80      take up next
F73B  85 59     STA $59       GCR-byte
F73D  A5 54     LDA $54       Set least sig. part (bit 0-3)
F73F  29 0F     AND #$0F      of 3rd byte to be converted
F741  AA        TAX           and determine the 5-bit GCR
F742  BD 7F F7  LDA $F77F,X   code to be adapted
F745  0A        ASL A         Set GCR-value in positions
F746  0A        ASL A         2-6 and Save as 2nd part
F747  29 7C     AND #$7C      of the 4th
F749  05 59     ORA $59       GCR-byte
F74B  85 59     STA $59       Save GCR-byte
F74D  A5 55     LDA $55       Get 4th bin. byte to be converted
F74F  29 F0     AND #$F0      and isolate most significant part
F751  4A        LSR A         (4-7)
F752  4A        LSR A         Half-byte in least sig. bytehalves
F753  4A        LSR A         shifted so that bytes can serve as
F754  4A        LSR A         pointers for the GCR-values
F755  AA        TAX           then get the binary byte's
F756  BD 7F F7  LDA $F77F,X   corresponding 5-bit-GCR-code
F759  6A        ROR A         First 3 bits of
F75A  66 5A     ROR $5A       GCR-value (position 0-2)
F75C  6A        ROR A         transferred to positions
F75D  66 5A     ROR $5A       5-7 of the last
F75F  6A        ROR A         GCR-value
F760  66 5A     ROR $5A       Carry the
F762  29 03     AND #$03      remaining 2 bits
F764  05 59     ORA $59       Combine with preceding GCR-value
F766  91 30     STA ($30),Y   and write to buffer
F768  C8        INY           Set buffer pointer to next byte
F769  D0 04     BNE $F76F     End of buffer reached?
F76B  A5 2F     LDA $2F       YES—Set pointer to data buffer
F76D  85 31     STA $31       again
F76F1 A5 55     LDA $55       Get last half-byte from last
F771  29 0F     AND #$0F      binary byte, and
F773  AA        TAX           save it
F774  BD 7F F7  LDA $F77F,X   Establish GCR-value, and
F777  05 5A     ORA $5A       combine with last GCR byte
F779  91 30     STA ($30),Y   Write byte to buffer
F77B  C8        INY           Set buffer pointer to next byte &
F77C  84 34     STY $34       save it
F77E  60        RTS           Return from this subroutine
----------------------------------------------------------------
[F6E1/F6EE/F707/F714/F72E/F742/F756/F774]
F77F  0A 0B 12 13 0E 0F 16 17   This table of 16 half-bytes
F787  09 19 1A 1B 0D 1D 1E 15   correspond to 5-bit-GCR-bytes
----------------------------------------------------------------
```

```
[9706/9BA3/9C20/F586/F69D/FCA7]
Buffer contents converted from binary to GCR-bytes
F78F  A9 00     LDA #$00       Lo-bytes of pmtrs set to null:
F791  85 30     STA $30        pointer to current GCR-buffer
F793  85 2E     STA $2E        pointer to current binary buffer
F795  85 36     STA $36        pointer to current buffer position
F797  A9 BB     LDA #$BB       Lo-byte for pointer set on
F799  85 34     STA $34          conditional buffer
F79B  85 50     STA $50        Flag for "buffer in GCR-code"
F79D  A5 31     LDA $31        Set pointer for current data
F79F  85 2F     STA $2F        buffer
F7A1  A9 01     LDA #$01       Turn pointer to conditional buffer
F7A3  85 31     STA $31        (high byte)
F7A5  A5 47     LDA $47        Identifier for data block
F7A7  85 52     STA $52        set as first char to be converted
F7A9  A4 36     LDY $36        Get buffer pointer
F7AB  B1 2E     LDA ($2E),Y    Get data byte from buffer and save
F7AD  85 53     STA $53        as 1st char to be converted
F7AF  C8        INY            Increment buffer pointer
F7B0  B1 2E     LDA ($2E),Y    Get next data byte and save as 2nd
F7B2  85 54     STA $54        byte to be converted
F7B4  C8        INY            Set buffer pointer to next char,
F7B5  B1 2E     LDA ($2E),Y    Get byte frm databuffer & save as
F7B7  85 55     STA $55        third byte to be converted
F7B9  C8        INY            Set buffer pntr to next byte, and
F7BA1 84 36     STY $36        save
F7BC  20 D0 F6  JSR $F6D0      4 bin.bytes convrted to 5 GCRbytes
F7BF  A4 36     LDY $36        Get buffer pointer again
F7C1  B1 2E     LDA ($2E),Y    Get next byte to be converted and
F7C3  85 52     STA $52        save in temporary storage
F7C5  C8        INY            Set buffer pointer to next char
F7C6  F0 11     BEQ $F7D9      End of interim buffers reached?
F7C8  B1 2E     LDA ($2E),Y    Get 2nd data byte for conversion,
F7CA  85 53     STA $53        and save it
F7CC  C8        INY            Increment buffer pointer
F7CD  B1 2E     LDA ($2E),Y    Get third byte for conversion and
F7CF  85 54     STA $54        store in GCR buffer
F7D1  C8        INY            Set buffer pointer to next byte
F7D2  B1 2E     LDA ($2E),Y    Get 4th byte for conversion and
F7D4  85 55     STA $55        save it
F7D6  C8        INY            Set buffer pointer to next char
```

```
F7D7  D0 E1     BNE $F7BA        Entire buffer converted?
F7D9¹ A5 3A     LDA $3A          Save data block
F7DB  85 53     STA $53          checksum
F7DD  A9 00     LDA #$00         and put fill characters in
F7DF  85 54     STA $54          the remainder of the
F7E1  85 55     STA $55          GCR work buffer
F7E3  4C D0 F6  JMP $F6D0        4 binary bytes to 5 GCR-values
----------------------------------------------------------------
[BF2D/F4A9/F4B8/F60A/F624/F64F/F8F4/F90E]  VGL. 98D9
5 GCR-bytes converted into 4 binary bytes
F7E6  A4 34     LDY $34          Get buffer pointer again
F7E8  B1 30     LDA ($30),Y      Get first byte from buffer and
F7EA  29 F8     AND #$F8         isolate the
F7EC  4A        LSR A            5-bit GCR value
F7ED  4A        LSR A            Then set up 8-bit value whereby
F7EE  4A        LSR A            the 5 GCR-bits take up
F7EF  85 56     STA $56          positions 0-4 , and save value
F7F1  B1 30     LDA ($30),Y      Get 2nd byte from buffer and
F7F3  29 07     AND #$07         isolate 3 bits of next GCR-byte,
F7F5  0A        ASL A            and move to bit positions
F7F6  0A        ASL A            0-4
F7F7  85 57     STA $57          Save resulting GCR-byte
F7F9  C8        INY              Set buffer pointer to next char
F7FA  D0 06     BNE $F802        End of buffer reached?
F7FC  A5 4E     LDA $4E          Set up pointer to current
F7FE  85 31     STA $31          data buffer
F800  A4 4F     LDY $4F          Pointer to buffer position
F802¹ B1 30     LDA ($30),Y      Get 2nd GCR-byte
F804  29 C0     AND #$C0         & remaining 2 bits of GCR-value
F806  2A        ROL A            and move to proper
F807  2A        ROL A            position
F808  2A        ROL A            (Bits 0-1)
F809  05 57     ORA $57          Combine 1st section
F80B  85 57     STA $57          Save GCR-value
F80D  B1 30     LDA ($30),Y      Get GCR-byte from buffer
F80F  29 3E     AND #$3E         Set up next GCR-value
F811  4A        LSR A            and set in output position
F812  85 58     STA $58          Save value
F814  B1 30     LDA ($30),Y      Get next GCR-byte
F816  29 01     AND #$01         & get 1ST part of next GCR-value
F818  0A        ASL A            Get
F819  0A        ASL A            bit in
F81A  0A        ASL A            position 4
F81B  0A        ASL A            of the byte
F81C  85 59     STA $59          and save value
F81E  C8        INY              Set pointer to next byte
F81F  B1 30     LDA ($30),Y      and get GCR-byte from buffer
```

```
F821   29 F0    AND #$F0       Get 2nd part of GCR-value
F823   4A       LSR A          and
F824   4A       LSR A          shift the lower
F825   4A       LSR A          half of the byte
F826   4A       LSR A          (positions 0-3)
F827   05 59    ORA $59        Combine with previous bits
F829   85 59    STA $59        and save GCR vlaue
F82B   B1 30    LDA ($30),Y    Get GCR-byte from buffer again
F82D   29 0F    AND #$0F       and then get the 1st four bits
F82F   0A       ASL A          of the next GCR-value
F830   85 5A    STA $5A        and save them
F832   C8       INY            Buffer pointer to next byte
F833   B1 30    LDA ($30),Y    Get GCR-byte from buffer
F835   29 80    AND #$80       and get last bit of
F837   18       CLC            preceding
F838   2A       ROL A          GCR-value
F839   2A       ROL A          Move bit to position 0
F83A   29 01    AND #$01       of byte and combine with
F83C   05 5A    ORA $5A        4 previous bits
F83E   85 5A    STA $5A        Save GCR-value
F840   B1 30    LDA ($30),Y    Get GCR-byte from buffer again
F842   29 7C    AND #$7C       Isolate GCR-value
F844   4A       LSR A          and shift postions 0-4
F845   4A       LSR A          of byte
F846   85 5B    STA $5B        Save value
F848   B1 30    LDA ($30),Y    Get GCR-byte again
F84A   29 03    AND #$03       and get 2 bits of the
F84C   0A       ASL A          next GCR-value
F84D   0A       ASL A          Shift bits in postions
F84E   0A       ASL A          3 and 4
F84F   85 5C    STA $5C        Save value
F851   C8       INY            Buffer pointer to next byte
F852   D0 06    BNE $F85A      End of buffer reached?
F854   A5 4E    LDA $4E        Turn buffer pointer to
F856   85 31    STA $31        current data buffer
F858   A4 4F    LDY $4F        Get position pointer again
F85A   B1 30    LDA ($30),Y    Read GCR-byte from buffer
F85C   29 E0    AND #$E0       and isolate remaining 3 bits from
F85E   2A       ROL A          previous GCR-values
F85F   2A       ROL A          Shift bits in positions
F860   2A       ROL A          0-2
F861   2A       ROL A          (using a carry)
F862   05 5C    ORA $5C        Combine previous 2 bits
F864   85 5C    STA $5C        Save pure GCR-value
F866   B1 30    LDA ($30),Y    Get byte from GCR-buffer
F868   29 1F    AND #$1F       Isolate last GCR-value
F86A   85 5D    STA $5D        and save it
```

```
F86C  C8        INY                 Buffer pointer to next byte,
F86D  84 34     STY $34             and save it
F86F  A6 56     LDX $56             Load 1st 5-bit-GCR-byte
F871  BD A0 F8  LDA $F8A0,X         and equivalent most sig. part
F874  A6 57     LDX $57             with least sig. part, by which
F876  1D C0 F8  ORA $F8C0,X         the 2nd GCR-byte declares,
F879  85 52     STA $52             combines & saves as binary bytes
F87B  A6 58     LDX $58             Load 3rd 5-bit GCRbyte and equiv.
F87D  BD A0 F8  LDA $F8A0,X         most sig. part with the least
F880  A6 59     LDX $59             sig. part, through which the 4th
F882  1D C0 F8  ORA $F8C0,X         GCR-byte will declare, combine &
F885  85 53     STA $53             save as binary bytes
F887  A6 5A     LDX $5A             Load 5th 5-bit-GCRbyte and equiv.
F889  BD A0 F8  LDA $F8A0,X         most sig. part with the least
F88C  A6 5B     LDX $5B             sig. part, by which
F88E  1D C0 F8  ORA $F8C0,X         the 6th GCR-byte will declare,
F891  85 54     STA $54             combine and save as binary bytes
F893  A6 5C     LDX $5C             Load 7th 5-bit-GCRbyte and equiv.
F895  BD A0 F8  LDA $F8A0,X         most sig. part with the
F898  A6 5D     LDX $5D             least sig. part,  by which
F89A  1D C0 F8  ORA $F8C0,X         the 8th GCR-byte will declare,
F89D  85 55     STA $55             combine and save as binary bytes
F89F  60        RTS                 Return from this subroutine
--------------------------------------------------------------------
Table of the most significant parts of GCR equivalents of binary
bytes; $FF means that this GCR value is undefined
F8A0  FF FF FF FF FF FF FF FF
F8A8  FF 80 00 10 FF C0 40 50
F8B0  FF FF 20 30 FF F0 60 70
F8B8  FF 90 A0 B0 FF D0 E0 FF
--------------------------------------------------------------------
Table of the least significant parts of GCR equivalents of binary
bytes; $FF means that this GCR value is undefined
F8C0  FF FF FF FF FF FF FF FF
F8C8  FF 08 00 01 FF 0C 04 05
F8D0  FF FF 02 03 FF 0F 06 07
F8D8  FF 09 0A 0B FF 0D 0E FF
--------------------------------------------------------------------
```

```
[BF24/F4ED]  cf. 9965
Convert status buffer from GCR to binary
F8E0  A9 00      LDA #$00     Reset pointer to
F8E2  85 34      STA $34      current GCR-byte
F8E4  85 2E      STA $2E      Clear pointer to target buffer
F8E6  85 36      STA $36      Pointer to current data position
F8E8  A9 01      LDA #$01     Set pointers $4E/$4F to
F8EA  85 4E      STA $4E      the beginning of the
F8EC  A9 BA      LDA #$BA     status buffer
F8EE  85 4F      STA $4F      from $01BB-$01FF
F8F0  A5 31      LDA $31      Set buffer pointer to value of
F8F2  85 2F      STA $2F      current data buffer
F8F4  20 E6 F7   JSR $F7E6    Convrt 5 GCRbytes to 4binarybytes
F8F7  A5 52      LDA $52      Set first converted byte as
F8F9  85 38      STA $38      header block identifier
F8FB  A4 36      LDY $36      Set pointer in buffer
F8FD  A5 53      LDA $53      Write second converted byte into
F8FF  91 2E      STA ($2E),Y  current data buffer
F901  C8         INY          Write third
F902  A5 54      LDA $54      converted byte
F904  91 2E      STA ($2E),Y  into current data buffer
F906  C8         INY          Write fourth
F907  A5 55      LDA $55      converted byte into
F909  91 2E      STA ($2E),Y  current data buffer
F90B  C8         INY          Set buffer pointer to next byte;
F90C[1] 84 36    STY $36      save it down
F90E  20 E6 F7   JSR $F7E6    Convrt 5 GCRbytes to 4binarybytes
F911  A4 36      LDY $36      Get buffer pointer again
F913  A5 52      LDA $52      Write first conveted byte
F915  91 2E      STA ($2E),Y  into current data buffer
F917  C8         INY          Set buffer pointer to next byte
F918  F0 11      BEQ $F92B    Data buffer full?
F91A  A5 53      LDA $53      NO—Write second converted byte
F91C  91 2E      STA ($2E),Y  into current data buffer
F91E  C8         INY          Write third converted
F91F  A5 54      LDA $54      byte into the
F921  91 2E      STA ($2E),Y  current data buffer
F923  C8         INY          Write fourth converted
F924  A5 55      LDA $55      byte into the
F926  91 2E      STA ($2E),Y  current data buffer
F928  C8         INY          Set buffer pointer to next byte
F929  D0 E1      BNE $F90C    Data buffer already full?
F92B[1] A5 53    LDA $53      YES—Then save second converted
F92D  85 3A      STA $3A      byte as checksum (parity)
F92F  A5 2F      LDA $2F      Prepare pointer  to
F931  85 31      STA $31      current data buffer
F933  60         RTS          Return from this subroutine
```

```
[972E/BF1B/F533]
Convert sector header into GCR-bytes
F934  A5 31     LDA $31        Take up pointer
F936  85 2F     STA $2F        to current data buffer
F938  A9 00     LDA #$00       Turn data pointer to
F93A  85 31     STA $31        header buffer
F93C  A9 24     LDA #$24       which begins
F93E  85 34     STA $34        at $24
F940  A5 39     LDA $39        Identifier for blockheader (8)
F942  85 52     STA $52        in temp storage for GCR-routine
F944  A5 1A     LDA $1A        Blockheader checksum in
F946  85 53     STA $53        temporary storage for GCR-routine
F948  A5 19     LDA $19        Data block sector number in
F94A  85 54     STA $54        temporary storage for GCR-routine
F94C  A5 18     LDA $18        Track number of data block
F94E  85 55     STA $55        in temp storage for GCR-Routine
F950  20 D0 F6  JSR $F6D0      Convrt 4binarybytes to 5 GCRbytes
F953  A5 17     LDA $17        2nd character of ID
F955  85 52     STA $52        in temp storage for GCR-Routine
F957  A5 16     LDA $16        First character of ID
F959  85 53     STA $53        in temp storage for GCR-Routine
F95B  A9 00     LDA #$00       Temporary storage for GCR-Routine
F95D  85 54     STA $54        filled with
F95F  85 55     STA $55        two empty spaces
F961  20 D0 F6  JSR $F6D0      Convrt 4binarybytes to 5 GCRbytes
F964  A5 2F     LDA $2F        Prep current
F966  85 31     STA $31        data buffer pointer
F968  60        RTS            Return from this subroutine
----------------------------------------------------------------
[BF12/F2D5/F390/F40D/F420/F4F8/F507/F553/F583/F6C7/FDA0/FDE2]cf99B5
End current job; prepare error return message
F969  A4 3F     LDY $3F        Buffer number of job
F96B  99 00 00  STA $0000,Y    Error message in command register
F96E  A5 50     LDA $50        Flag for GCR-format
F970  F0 03     BEQ $F975      Data still in GCR-code?
F972  20 F2 F5  JSR $F5F2      YES—Convert GCR-data
F975  20 8F F9  JSR $F98F      Drive motor off
F978  A6 49     LDX $49        Temporary storage for Stack
F97A  9A        TXS            Reset stack to read whether
F97B  4C BE F2  JMP $F2BE      a new job is there
----------------------------------------------------------------
[92F5/F2DF]
Drive motor on; wait until motor is constantly on
F97E  A9 A0     LDA #$A0       Set 'Motor runs on' flag
F980  85 20     STA $20        as drive status
F982  AD 00 1C  LDA $1C00      Get control register
F985  09 04     ORA #$04       and set motor bit (Bit2)
```

```
F987  8D 00 1C   STA $1C00      to 'motor on' (=1)
F98A  A9 32      LDA #$32       Set counter for 0.8 / 0.4 secs.
F98C  85 48      STA $48        delay time
F98E  60         RTS            Return from this subroutine
-----------------------------------------------------------------
[86CF/98C1/F975]
Drive  motor off
F98F  A6 3E      LDX $3E        Current drive
F991  A5 20      LDA $20        Get drive status and switch off
F993  09 10      ORA #$10       flag for
F995  85 20      STA $20        'motor off'
F997  4C 2B A6   JMP $A62B      Set runtime counter
-----------------------------------------------------------------
F99A  EA         NOP            unused [1541 ROM
F99B  EA         NOP            modification]
-----------------------------------------------------------------
[BF6F/F2E6/F2F0/F2F6/F31D/F48A/FAF2/FAFD/FD93/FDD8]
Main disk control routine
F99C  AD 07 1C   LDA $1C07      Timer w/# of time cycles(hi-byte)
F99F  8D 05 1C   STA $1C05      set until next IRQ
F9A2  AD 00 1C   LDA $1C00      Check status of light box for
F9A5  29 10      AND #$10       write-protect on disk
F9A7  C5 1E      CMP $1E        or disk change
F9A9  85 1E      STA $1E        Save write-protect status
F9AB  4C 34 A6   JMP $A634      Switch motor
-----------------------------------------------------------------
F9AE  EA         NOP            Unused
F9AF  EA         NOP            section, due to modification
F9B0  EA         NOP            of 1541-ROM
-----------------------------------------------------------------
[A657]
Control head
F9B1  AD FE 02   LDA $02FE      Status flag for step-motor
F9B4  F0 15      BEQ $F9CB      Is the head on the chosen track?
F9B6  C9 02      CMP #$02       NO—Is the head even positioned
F9B8  D0 07      BNE $F9C1      on the chosen track area?
F9BA  A9 00      LDA #$00       YES—Set 'head on
F9BC  8D FE 02   STA $02FE      track' flag
F9BF  F0 0A      BEQ $F9CB      Jump to $F9CB
F9C1[1] 85 4A    STA $4A        Counter for number of half-steps
F9C3  A9 02      LDA #$02       Set head status flag to
F9C5  8D FE 02   STA $02FE      'Head on track' and move a
F9C8  4C 2E FA   JMP $FA2E      1/2 step along on chosen track
F9CB[2] A6 3E    LDX $3E        Drive motor status
F9CD  30 07      BMI $F9D6      Motor running?
F9CF  A5 20      LDA $20        YES—Test drive status flag
F9D1  A8         TAY            Save it down
```

```
F9D2   C9 20      CMP #$20       Motor on and off
F9D4   D0 03      BNE $F9D9      Constant turning number?
F9D6²  4C BE FA   JMP $FABE      Circuitry initialization
F9D9¹  C6 48      DEC $48        Delay counter for motor run;
F9DB   D0 1D      BNE $F9FA      motors at turn number?
F9DD   98         TYA            YES—Get drive status again
F9DE   10 04      BPL $F9E4      Drive ready?
F9E0   29 7F      AND #$7F       NO—Clear flag for motor
F9E2   85 20      STA $20        pause; note this
F9E4¹  29 10      AND #$10       Test bit for 'motor runs'
F9E6   F0 12      BEQ $F9FA      Active?
F9E8   C6 35      DEC $35        YES—Number of jobloop call
F9EA   D0 0E      BNE $F9FA      Need the step motor again?
F9EC   EA         NOP            NO—Drive motor
F9ED   20 70 87   JSR $8770      off
F9F0   A9 FF      LDA #$FF       Clear 'drive active'
F9F2   85 3E      STA $3E        flag
F9F4   A9 00      LDA #$00       Clear drive status
F9F6   85 20      STA $20        flag
F9F8   F0 DC      BEQ $F9D6      Jump to $F9D6
F9FA³  98         TYA            Drive status flag
F9FB   29 40      AND #$40       Isolate flag for stepper status
F9FD   D0 03      BNE $FA02      Should head be moved?
F9FF   4C BE FA   JMP $FABE      NO—Jump to $FABE
FA02¹  6C 62 00   JMP ($0062)    YES—Goto current stepper routine

Possible routine calls:

Initialize head control           $FA05
Slow head movement                $FA3B
End head movement                 $FA4E
Initialize fast head movement     $FA7B
Fast head movement                $FA97
Slow down fast head movement      $FAA5

-------------------------------------------------------------------
[Jump over FA02]
Initialization routine for head movement
FA05   A5 4A      LDA $4A        Number of half-steps up to track
FA07   10 05      BPL $FA0E      Should the head move in?
FA09   49 FF      EOR #$FF       YES—The supply the
FA0B   18         CLC            step size with
FA0C   69 01      ADC #$01       positive leading character
FA0E¹  C5 64      CMP $64        Test for 'Head at track 0'
FA10   B0 0A      BCS $FA1C      Fast head movement?
FA12   A9 3B      LDA #$3B       NO—Call  $FA02
FA14   85 62      STA $62        Set slow head movement routine,
```

```
FA16  A9 FA      LDA #$FA     where pointers $62/63 are turned
FA18  85 63      STA $63      to $FA3B
FA1A  D0 12      BNE $FA2E    Jump to $FA2E
FA1C1 E5 5E      SBC $5E      Go to # of steps for motor and
FA1E  E5 5E      SBC $5E      slow motor(by 4)by total steps;
FA20  85 61      STA $61      save it
FA22  A5 5E      LDA $5E      Set pointer for number of
FA24  85 60      STA $60      running steps
FA26  A9 7B      LDA #$7B     Call for routine in $FA02
FA28  85 62      STA $62      for fast head movement;
FA2A  A9 FA      LDA #$FA     set pointers in
FA2C  85 63      STA $63      $62/$63 to $FA7B
FA2E4 A5 4A      LDA $4A      Step pointer
FA30  10 31      BPL $FA63    Inward movement?
FA32  4C 36 FF   JMP $FF36    YES—Control stepper motor
-------------------------------------------------------------
FA35  EA ...     NOP          unused
FA37  ... EA     NOP          ROM area
-------------------------------------------------------------
[FF7F]
FA38  4C 69 FA   JMP $FA69    Control stepper
-------------------------------------------------------------
[Jump by FA02]
Execute slow head movement for a short distance
FA3B  A5 4A      LDA $4A      Step counter for # of half-steps
FA3D  D0 EF      BNE $FA2E    Target spur reached?
FA3F  A9 4E      LDA #$4E     Call in $FA02 for routine
FA41  85 62      STA $62      to end head movement
FA43  A9 FA      LDA #$FA     set, in which the pointers
FA45  85 63      STA $63      $62/$63 are turned to $FA7B
FA47  A9 05      LDA #$05     Fifth half-step set to stop
FA49  85 60      STA $60      head
FA4B  4C BE FA   JMP $FABE    Prep byte ready flag
-------------------------------------------------------------
[Originates at FA02]
End of head movement
FA4E  C6 60      DEC $60      Number of steps to brake head
FA50  D0 6C      BNE $FABE    Braking procedure executed?
FA52  A5 20      LDA $20      Drive status flag
FA54  29 BF      AND #$BF     Reset bitflag for
FA56  85 20      STA $20      head in motion
FA58  A9 05      LDA #$05     Call in $FA02 for routine to
FA5A  85 62      STA $62      initialize head movement
FA5C  A9 FA      LDA #$FA     Set, in which the pointers in
FA5E  85 63      STA $63      $62/$63 are set at $FA05
FA60  4C BE FA   JMP $FABE    Prep Byte Ready Flag
-------------------------------------------------------------
```

```
[FA30]
Control stepmotor
FA63   C6 4A      DEC $4A       Number of half-track steps
FA65   AE 00 1C   LDX $1C00     Control port for stepper motor
FA68   E8         INX           Move head outward in which the
FA69[1] 8A        TXA           stepper bits 0 & 1 will be
FA6A   29 03      AND #$03      counted outwards: isolate and
FA6C   85 4B      STA $4B       save control bits
FA6E   AD 00 1C   LDA $1C00     Get drive control reg. and clear
FA71   29 FC      AND #$FC      stepper motor bits
FA73   05 4B      ORA $4B       Conbine previously computed bits
FA75   8D 00 1C   STA $1C00     and control stepper motor
FA78   4C BE FA   JMP $FABE     Prepare byte ready flag
--------------------------------------------------------------
[Originates at FA02]
Set up fast head movement and move  head
FA7B   38         SEC           Time constant until
FA7C   AD 07 1C   LDA $1C07     next call
FA7F   E5 5F      SBC $5F       to decrement driving constant(4);
FA81   8D 05 1C   STA $1C05     this conveys stepper impulse
FA84   C6 60      DEC $60       to traveler
FA86   D0 0C      BNE $FA94     Four driving impulses given?
FA88   A5 5E      LDA $5E       YES—Set counter for later
FA8A   85 60      STA $60       braking
FA8C   A9 97      LDA #$97      Call $FA02 to routine
FA8E   85 62      STA $62       to set fast head move-
FA90   A9 FA      LDA #$FA      ment, in which the pointers of
FA92   85 63      STA $63       $62/$63 are set to $FA97
FA94[4] 4C 2E FA  JMP $FA2E     Move head
--------------------------------------------------------------
[Originates at FA02]
Execute fast head movement
FA97   C6 61      DEC $61       Half-step counter
FA99   D0 F9      BNE $FA94     Reached target?
FA9B   A9 A5      LDA #$A5      YES—Set call in $FA02 to routine
FA9D   85 62      STA $62       for head braking, in
FA9F   A9 FA      LDA #$FA      which pointers
FAA1   85 63      STA $63       $62/$63 are set to $FAA5
FAA3   D0 EF      BNE $FA94     Jump to $FA94
--------------------------------------------------------------
[Originates at FA02]
Braking head after fast movement
FAA5   AD 07 1C   LDA $1C07     Increase time constant until
FAA8   18         CLC           next call, which will slow down
FAA9   65 5F      ADC $5F       stepper impulses,  to prevent a
FAAB   8D 05 1C   STA $1C05     'track overflow'
FAAE   C6 60      DEC $60       Counter for braking impulse
```

```
FAB0   D0 E2      BNE $FA94       Already stopped?
FAB2   A9 4E      LDA #$4E        YES—Set call in $FA02 to
FAB4   85 62      STA $62         end head transport routine,
FAB6   A9 FA      LDA #$FA        in which pointers
FAB8   85 63      STA $63         $62/$63 are turned to $FA4E
FABA   A9 05      LDA #$05        Reset number of
FABC   85 60      STA $60         braking impulses
------------------------------------------------------------------
[F9D6/F9FF/FA4B/FA50/FA60/FA78/FF74]
FABE   AD 0C 1C   LDA $1C0C       Initialize read/write circuitry
FAC1   29 FD      AND #$FD        in which bit 1 (Byte Ready Flag)
FAC3   8D 0C 1C   STA $1C0C       will be reset
FAC6   60         RTS             Return from this subroutine
------------------------------------------------------------------
Format diskette in 1541 format
Original follows at ($0600), where you can put your own programs
FAC7   A5 51      LDA $51         Current track number
FAC9   10 2A      BPL $FAF5       Format procedure already started?
FACB   A6 3D      LDX $3D         NO—Get current drive number, and
FACD   A9 60      LDA #$60        set head movement flag for drive
FACF   95 20      STA $20,X       status flag (Bit 6/5)
FAD1   A9 01      LDA #$01        Track 11 as disk's start track
FAD3   95 22      STA $22,X       controller --
FAD5   85 51      STA $51         save it
FAD7   A9 A4      LDA #$A4        Move head 46 tracks(til strking)
FAD9   85 4A      STA $4A         outward
FADB   AD 00 1C   LDA $1C00       Clear controlbits f/stepper motor
FADE   29 FC      AND #$FC        and give to
FAE0   8D 00 1C   STA $1C00       stepper
FAE3   A9 0A      LDA #$0A        Set maximum number of format
FAE5   8D 20 06   STA $0620       tries
FAE8   A9 A0      LDA #$A0        Set starting value f/named track
FAEA   8D 21 06   STA $0621       capacity in $0621/$0622 to $0FA0
FAED   A9 0F      LDA #$0F        (which is equal to
FAEF   8D 22 06   STA $0622       4000 bytes capacity)
FAF2   4C 9C F9   JMP $F99C       Move head on track
FAF5[1] A0 00     LDY #$00        Compare current track number with
FAF7   D1 32      CMP ($32),Y     number in temporary storage
FAF9   F0 05      BEQ $FB00       Still same track being worked on?
FAFB   91 32      STA ($32),Y     NO—Get current track number
FAFD   4C 9C F9   JMP $F99C       Move head to new track
FB00[1] AD 00 1C  LDA $1C00       Get control register, and
FB03   29 10      AND #$10        test for write protect (Bit4)
FB05   D0 05      BNE $FB0C       Write protect on hand?
FB07   A9 08      LDA #$08        YES—Display
FB09   4C D3 FD   JMP $FDD3       '26 Write Protect On' error msg.
FB0C[2] 20 A3 FD  JSR $FDA3       Write $FF to entire track
```

```
FB0F    20 C3 FD   JSR $FDC3    Fill track capacity w/ $FF and
FB12    A9 55      LDA #$55     write in the same number of
FB14    8D 01 1C   STA $1C01    $55 bytes
FB17    20 C3 FD   JSR $FDC3    Capacity marked in $0621/$622
FB1A    20 00 FE   JSR $FE00    Switch head to Read mode
FB1D    20 56 F5   JSR $F556    Wait for first $FF byte (Sync)
FB20    A9 40      LDA #$40     Run of timer 1 will
FB22    0D 0B 18   ORA $180B    produce an impulse
FB25    8D 0B 18   STA $180B    on PB7 (ATN-input)
FB28    A9 62      LDA #$62     Timer 1 is programmed
FB2A    8D 06 18   STA $1806    for a runtime of
FB2D    A9 00      LDA #$00     62
FB2F    8D 07 18   STA $1807    impulses
FB32    8D 05 18   STA $1805    Start timer 1
FB35    A0 00      LDY #$00     Clear
FB37    A2 00      LDX #$00     counter
FB39[1] 2C 00 1C   BIT $1C00    Test sync-flag
FB3C    30 FB      BMI $FB39    Wait until sync-signal is gone
FB3E[1] 2C 00 1C   BIT $1C00    Check sync-flag
FB41    10 FB      BPL $FB3E    Wait until sync-range comes again
FB43[2] AD 04 18   LDA $1804    Get curr countr state from timer1
FB46[1] 2C 00 1C   BIT $1C00    Test sync-flag
FB49    10 11      BPL $FB5C    Is sync range now past?
FB4B    AD 0D 18   LDA $180D    NO—Get interrupt flags
FB4E    0A         ASL A        and test 'Timer 1 running' flag
FB4F    10 F5      BPL $FB46    Time up?
FB51    E8         INX          YES—Increment timer
FB52    D0 EF      BNE $FB43    Run into a transfer?
FB54    C8         INY          YES—Correct high-byte of counter
FB55    D0 EC      BNE $FB43    Timer overrun?
FB57    A9 02      LDA #$02     YES—Display
FB59    4C D3 FD   JMP $FDD3    '20 Read Error' message
FB5C[1] 86 71      STX $71      Save number of
FB5E    84 72      STY $72      $55 bytes
FB60    A2 00      LDX #$00     Clear register for next
FB62    A0 00      LDY #$00     count
FB64[2] AD 04 18   LDA $1804    Get counter state of timer 1
FB67[1] 2C 00 1C   BIT $1C00    Check sync-flag
FB6A    30 11      BMI $FB7D    Is head over sync range?
FB6C    AD 0D 18   LDA $180D    NO—Get interrupt flag and
FB6F    0A         ASL A        test 'Timer 1 running' flag
FB70    10 F5      BPL $FB67    Time up?
FB72    E8         INX          YES—Increment counter
FB73    D0 EF      BNE $FB64    Reached a transfer?
FB75    C8         INY          YES—Correct high-byte of counter
FB76    D0 EC      BNE $FB64    Counter overflow?
FB78    A9 02      LDA #$02     YES—Display
```

```
FB7A   4C D3 FD   JMP $FDD3    '20 Read Error' message
FB7D1  38         SEC          Calculate difference
FB7E   8A         TXA          between $55 range and
FB7F   E5 71      SBC $71      the $FF range;
FB81   AA         TAX          save in
FB82   85 70      STA $70      pointers $70/$71 for
FB84   98         TYA          determining
FB85   E5 72      SBC $72      real track
FB87   A8         TAY          capacity
FB88   85 71      STA $71      (Take $71/72 frm X/Y & in $70/71)
FB8A   10 0B      BPL $FB97    Is value negative?
FB8C   49 FF      EOR #$FF     YES—Draw up 2nd complement of
FB8E   A8         TAY          values
FB8F   8A         TXA          (give absolute value)
FB90   49 FF      EOR #$FF     Complement low-byte
FB92   AA         TAX          and save it
FB93   E8         INX          Design 2nd complement
FB94   D0 01      BNE $FB97    is one a transfer?
FB96   C8         INY          YES—Correct and get
FB972  98         TYA          high-byte
FB98   D0 04      BNE $FB9E    Is value in X/Y less than 256?
FB9A   E0 04      CPX #$04     YES—Compare low-byte (X) with 4
FB9C   90 18      BCC $FBB6    Track capacity same as 4 bytes?
FB9E1  06 70      ASL $70      NO—Double track capacity
FBA0   26 71      ROL $71      value
FBA2   18         CLC          and calculate for track capacity
FBA3   A5 70      LDA $70      Get low-byte and add to
FBA5   6D 21 06   ADC $0621    awaited value
FBA8   8D 21 06   STA $0621    Save newly-awaited value
FBAB   A5 71      LDA $71      Get high-byte and  add
FBAD   6D 22 06   ADC $0622    to
FBB0   8D 22 06   STA $0622    awaited value
FBB3   4C 0C FB   JMP $FB0C    Determine track capacity again
FBB61  A2 00      LDX #$00     Clear
FBB8   A0 00      LDY #$00     counter
FBBA   B8         CLV          Prepare 'byte ready' flag
FBBB3  AD 00 1C   LDA $1C00    Test flag for sync-signal
FBBE   10 0E      BPL $FBCE    Is head over sync range?
FBC0   50 F9      BVC $FBBB    YES—Wait for next byte
FBC2   B8         CLV          Prep 'Byte Ready'
FBC3   E8         INX          Increment counter
FBC4   D0 F5      BNE $FBBB    Is there a transfer occurring?
FBC6   C8         INY          YES—Correct high-byte of counter
FBC7   D0 F2      BNE $FBBB    Is counter overflowing?
FBC9   A9 03      LDA #$03     YES—Set error #:'Sync not found'
FBCB   4C D3 FD   JMP $FDD3    and eventually re-test
FBCE1  8A         TXA          Double counter,put in $0625/$0624
```

```
FBCF   0A         ASL A       Double and save
FBD0   8D 25 06   STA $0625   low-byte
FBD3   98         TYA         Get high-byte and
FBD4   2A         ROL A       save as two
FBD5   8D 24 06   STA $0624   values
FBD8   A9 BF      LDA #$BF    Flag for 'Run from Timer 1'
FBDA   2D 0B 18   AND $180B   Get interrupt flag and
FBDD   8D 0B 18   STA $180B   reset flag
FBE0   A9 66      LDA #$66    # of bytes f/every sector needed
FBE2   8D 26 06   STA $0626   in addition to the 256 data
FBE5   A6 43      LDX $43     Number of sectors
FBE7   A0 00      LDY #$00    Index value f/# of 256byte blocks
FBE9   98         TYA         Start value for surplus calc.s(0)
FBEA1  18         CLC         and with it, calculate sector
FBEB   6D 26 06   ADC $0626   excess
FBEE   90 01      BCC $FBF1   Are 256 more bytes needed?
FBF0   C8         INY         Index raised by 256 bytes
FBF11  C8         INY         Index raised by 256 bytes
FBF2   CA         DEX         Compute next sector
FBF3   D0 F5      BNE $FBEA   All sectors considered?
FBF5   49 FF      EOR #$FF    YES—Compute 2nd complement
FBF7   38         SEC         (negative value) of remaining
FBF8   69 00      ADC #$00    necessary bytes
FBFA   18         CLC         and subtract from total
FBFB   6D 25 06   ADC $0625   capacity (add negative value)
FBFE   B0 03      BCS $FC03   Need to borrow?
FC00   CE 24 06   DEC $0624   YES—Correct high-byte
FC031  AA         TAX         Save low-byte of capacity
FC04   98         TYA         Get # of necessary 256byte blocks
FC05   49 FF      EOR #$FF    and draw up 2nd complement
FC07   38         SEC         (negative value)
FC08   69 00      ADC #$00    from that
FC0A   18         CLC         Subtract # of neces 256byteblocks
FC0B   6D 24 06   ADC $0624   from total capacity
FC0E   10 05      BPL $FC15   Sufficient track capacity?
FC10   A9 04      LDA #$04    NO—Display 'Block Not Found'
FC12   4C D3 FD   JMP $FDD3   error message
FC151  A8         TAY         Get number of remaining
FC16   8A         TXA         bytes
FC17   A2 00      LDX #$00    Counter for number of blank bytes
FC191  38         SEC         Number of bytes remaining
FC1A   E5 43      SBC $43     divided by number of sectors,
FC1C   B0 03      BCS $FC21   in which sector # will be divided
FC1E   88         DEY         by the empty bytes
FC1F   30 03      BMI $FC24   X counts as often as is possible
FC211  E8         INX         Increment number of blank bytes
FC22   D0 F5      BNE $FC19   Jump to $FC19
```

```
FC24[1] 8E 26 06   STX $0626     save number of blank bytes
FC27    E0 04      CPX #$04      and compare with 4 bytes
FC29    B0 05      BCS $FC30     Is skip smaller?
FC2B    A9 05      LDA #$05      YES—Display
FC2D    4C D3 FD   JMP $FDD3     '23 Read Error' message
FC30[1] 18         CLC           Add number of sectors
FC31    65 43      ADC $43       to track and
FC33    8D 27 06   STA $0627     save result
FC36    A9 00      LDA #$00      Reset counter for
FC38    8D 28 06   STA $0628     sectors written up
FC3B    A0 00      LDY #$00      Clear pointrs for blockheader set
FC3D    A6 3D      LDX $3D       up in buffer 1
FC3F[1] A5 39      LDA $39       Write blockheader identifier (8)
FC41    99 00 03   STA $0300,Y   into blockheader
FC44    C8         INY           Set pointer to next position
FC45    C8         INY           Jump over to checksum byte
FC46    AD 28 06   LDA $0628     Write number of current sector
FC49    99 00 03   STA $0300,Y   in blockheader
FC4C    C8         INY           Set pointer to next position
FC4D    A5 51      LDA $51       Take up number of current track
FC4F    99 00 03   STA $0300,Y   in blockheader
FC52    C8         INY           Set pointer to next position
FC53    B5 13      LDA $13,X     Write second ID character
FC55    99 00 03   STA $0300,Y   in blockheader
FC58    C8         INY           Set pointer to next position
FC59    B5 12      LDA $12,X     Transfer first ID character
FC5B    99 00 03   STA $0300,Y   to blockheader
FC5E    C8         INY           Set pointer to next position
FC5F    A9 0F      LDA #$0F      Write $0F (15)
FC61    99 00 03   STA $0300,Y   twice to fill
FC64    C8         INY           in the
FC65    99 00 03   STA $0300,Y   blockheader in
FC68    C8         INY           the buffer
FC69    A9 00      LDA #$00      Checksum for:
FC6B    59 FA 02   EOR $02FA,Y   Track number
FC6E    59 FB 02   EOR $02FB,Y   Sector number
FC71    59 FC 02   EOR $02FC,Y   Second ID-char.
FC74    59 FD 02   EOR $02FD,Y   First ID-char.
FC77    99 F9 02   STA $02F9,Y   Compute and set into blockheader
FC7A    EE 28 06   INC $0628     Set countr for current sector #
FC7D    AD 28 06   LDA $0628     to next sector; compare with
FC80    C5 43      CMP $43       value for max. sector number
FC82    90 BB      BCC $FC3F     All sectors covered?
FC84    98         TYA           YES—Keep pointer at
FC85    48         PHA           current buffer position
FC86    E8         INX            (1)
FC87    8A         TXA           Set up data block;
```

```
FC88¹ 9D 00 05   STA $0500,X    Write to buffer 1
FC8B  E8         INX            Set pointer to next byte
FC8C  D0 FA      BNE $FC88      Buffer full?
FC8E  A9 03      LDA #$03       YES—Set address $0300 as current
FC90  85 31      STA $31        buffer address
FC92  20 30 FE   JSR $FE30      Convrt buffer contents to GCRcode
FC95  68         PLA            Re-rig previous buffer position
FC96  A8         TAY            and set pointer to
FC97  88         DEY            start of blockheader
FC98  20 E5 FD   JSR $FDE5      Move status buffer contents to
FC9B  20 F5 FD   JSR $FDF5      buffer at $0300
FC9E  A9 05      LDA #$05       Set $0500 as curent
FCA0  85 31      STA $31        buffer address
FCA2  20 E9 F5   JSR $F5E9      Compute data block checksum and
FCA5  85 3A      STA $3A        save it
FCA7  20 8F F7   JSR $F78F      Change data block into GCR code
FCAA  A9 00      LDA #$00       Initialize pointer to  current
FCAC  85 32      STA $32        blockheader
FCAE  20 0E FE   JSR $FE0E      Clear track with $55
FCB1¹ A9 FF      LDA #$FF       Give identifier for sync-marking
FCB3  8D 01 1C   STA $1C01      to write head
FCB6  A2 05      LDX #$05       Number of sync-bytes
FCB8² 50 FE      BVC $FCB8      Wait for 'Byte Ready'
FCBA  B8         CLV            Prep 'Byte Ready' flag
FCBB  CA         DEX            Decrement counter
FCBC  D0 FA      BNE $FCB8      All sync-bytes already on Disk?
FCBE  A2 0A      LDX #$0A       Blockheader length
FCC0  A4 32      LDY $32        Pointer in position in buffer
FCC2² 50 FE      BVC $FCC2      Write circuitry ready?
FCC4  B8         CLV            YES—Set up flag again
FCC5  B9 00 03   LDA $0300,Y    Get GCR-bytes from buffer --
FCC8  8D 01 1C   STA $1C01      transfer to write head
FCCB  C8         INY            Buffer pointer to next character
FCCC  CA         DEX            # of chars. yet to be written
FCCD  D0 F3      BNE $FCC2      Header already written?
FCCF  A2 09      LDX #$09       YES—Write in spaces between
FCD1² 50 FE      BVC $FCD1      block-header and datablock
FCD3  B8         CLV            with fill values
FCD4  A9 55      LDA #$55       ($55)
FCD6  8D 01 1C   STA $1C01      Send byte over write head
FCD9  CA         DEX            Counter for number of fillbytes
FCDA  D0 F5      BNE $FCD1      Blanks aleady written?
FCDC  A9 FF      LDA #$FF       Write sync-mark for
FCDE  A2 05      LDX #$05       data blockheader to diskette
FCE0² 50 FE      BVC $FCE0      Write circuitry ready?
FCE2  B8         CLV            YES—Flag set again
FCE3  8D 01 1C   STA $1C01      Sync-byte to write circuitry
```

```
FCE6   CA          DEX              Counter for number of sync-bytes
FCE7   D0 F7       BNE $FCE0        Sync-marking already written?
FCE9   A2 BB       LDX #$BB         Pointer to start of temp. buffer
FCEB2  50 FE       BVC $FCEB        Write circuitry ready?
FCED   B8          CLV              YES—Prep 'Byte Ready' flag
FCEE   BD 00 01    LDA $0100,X      Get byte from buffer and
FCF1   8D 01 1C    STA $1C01        write to diskette
FCF4   E8          INX              Buffer pointer to next byte
FCF5   D0 F4       BNE $FCEB        Buffer written up?
FCF7   A0 00       LDY #$00         YES—Buffer pointer to data buffer
FCF92  50 FE       BVC $FCF9        Write circuitry ready?
FCFB   B8          CLV              YES—Prepare 'Byte Ready' flag
FCFC   B1 30       LDA ($30),Y      Write byte to diskette
FCFE   8D 01 1C    STA $1C01        from buffer
FD01   C8          INY              Pointer to next char in buffer
FD02   D0 F5       BNE $FCF9        Is entire buffer written already?
FD04   A9 55       LDA #$55         Fill space between 2 data blocks
FD06   AE 26 06    LDX $0626        Number of bytes per space
FD092  50 FE       BVC $FD09        Write circuitry ready?
FD0B   B8          CLV              YES—Reset flag
FD0C   8D 01 1C    STA $1C01        $55 to read head
FD0F   CA          DEX              Counter foor number of fillbytes
FD10   D0 F7       BNE $FD09        Blanks already written in?
FD12   A5 32       LDA $32          Buffer pointer (to header
FD14   18          CLC              position of next blockheader) --
FD15   69 0A       ADC #$0A         set and save
FD17   85 32       STA $32          this pointer
FD19   CE 28 06    DEC $0628        Draw up number of next sector
FD1C   D0 93       BNE $FCB1        All sectors already written?
FD1E1  50 FE       BVC $FD1E        YES—Wait for next byte
FD20   B8          CLV              Prep 'Byte Ready' flag
FD211  50 FE       BVC $FD21        Wait for  next byte
FD23   B8          CLV              Reset 'Byte Ready'
FD24   20 00 FE    JSR $FE00        —switch to read mode
FD27   A9 C8       LDA #$C8         Set number of read attempts
FD29   8D 23 06    STA $0623        (200)
FD2C   A9 00       LDA #$00         Set buffer pointer $30/$31
FD2E   85 30       STA $30          buffer 1
FD30   A9 03       LDA #$03         ($0300-$03FF)
FD32   85 31       STA $31          ($0300-$03FF)
FD34   A5 43       LDA $43          Save number of sectors
FD36   8D 28 06    STA $0628        per track
FD391  20 56 F5    JSR $F556        Wait for sync-marking
FD3C   A2 0A       LDX #$0A         Number of bytes in blockheader
FD3E   A0 00       LDY #$00         Clear buffer pointer
FD402  50 FE       BVC $FD40        Read circuitry ready?
FD42   B8          CLV              YES—Get flag ready
```

```
FD43   AD 01 1C   LDA $1C01      Read byte from diskette and
FD46   D1 30      CMP ($30),Y    compare with buffer
FD48   D0 0E      BNE $FD58      Blockheader being sought?
FD4A   C8         INY            Set pointer to nextbyte of
FD4B   CA         DEX            header
FD4C   D0 F2      BNE $FD40      Last byte of header compared?
FD4E   18         CLC            Set buffer address to
FD4F   A5 30      LDA $30        next blockheader
FD51   69 0A      ADC #$0A       in
FD53   85 30      STA $30        buffer memory
FD55   4C 62 FD   JMP $FD62      again
FD58[3] CE 23 06  DEC $0623      Number of read searches
FD5B   D0 CF      BNE $FD2C      Last search?
FD5D   A9 06      LDA #$06       YES—Display
FD5F   4C D3 FD   JMP $FDD3      '24 Read Error' message
FD62[1] 20 56 F5  JSR $F556      Wait f/syncmarking of data blocks
FD65   A0 BB      LDY #$BB       Set buffer pointer to temp.buffer
FD67[2] 50 FE     BVC $FD67      Read circuitry ready?
FD69   B8         CLV            YES—Reset Byte Ready Flag
FD6A   AD 01 1C   LDA $1C01      Compare byte from diskette
FD6D   D9 00 01   CMP $0100,Y    with buffer contents
FD70   D0 E6      BNE $FD58      Positive comparison?
FD72   C8         INY            YES—Buffer pointer to next byte
FD73   D0 F2      BNE $FD67      Entire buffer already compared?
FD75   A2 FC      LDX #$FC       YES—Counter f/ data buffer bytes
FD77[2] 50 FE     BVC $FD77      Read circuitry ready?
FD79   B8         CLV            YES—Set Byte Ready flag back
FD7A   AD 01 1C   LDA $1C01      Read byte from diskette and
FD7D   D9 00 05   CMP $0500,Y    compare with data buffer
FD80   D0 D6      BNE $FD58      Positive comparison?
FD82   C8         INY            YES—Set pointer to next
FD83   CA         DEX            byte
FD84   D0 F1      BNE $FD77      Last character of buffer
FD86   CE 28 06   DEC $0628      Number of sectors-1 of track
FD89   D0 AE      BNE $FD39      All sectors tested?
FD8B   E6 51      INC $51        YES—Increment track # counter
FD8D   A5 51      LDA $51        Set and save track; compare
FD8F   C9 24      CMP #$24       with max. number of tracks
FD91   B0 03      BCS $FD96      Reached track 35?
FD93   4C 9C F9   JMP $F99C      NO—Continue formatting
FD96[1] A9 FF     LDA #$FF       Set flag to
FD98   85 51      STA $51        end formatting
FD9A   A9 00      LDA #$00       Clear 'Buffer in GCR-Code'
FD9C   85 50      STA $50        flag
FD9E   A9 01      LDA #$01       Display 'ok' message;
FDA0   4C 69 F9   JMP $F969      End of formatting
-------------------------------------------------------------
```

```
[FB0C]
Write track with $FF
FDA3   AD 0C 1C   LDA $1C0C       Switch head circuitry
FDA6   29 1F      AND #$1F        in PCR-Register to
FDA8   09 C0      ORA #$C0        write
FDAA   8D 0C 1C   STA $1C0C       mode (CB2 = 0)
FDAD   A9 FF      LDA #$FF        Switch read/write head port
FDAF   8D 03 1C   STA $1C03       for output
FDB2   8D 01 1C   STA $1C01       Write $FF
FDB5   A2 28      LDX #$28        Set counter in CPU-Register
FDB7   A0 00      LDY #$00        to 10240
FDB9³  50 FE      BVC $FDB9       Wait for Byte Ready
FDBB   B8         CLV             Byte Ready Flag prepared
FDBC   88         DEY             Low-byte of counter
FDBD   D0 FA      BNE $FDB9       Executed once until null?
FDBF   CA         DEX             YES—Then decremnt hi-byte/counter
FDC0   D0 F7      BNE $FDB9       Already written 10240 times?
FDC2   60         RTS             YES—Return from this subroutine
-------------------------------------------------------------------
[FB0F/FB17]
($0621/$0622) times—wait on 'Byte Ready' signal
FDC3   AE 21 06   LDX $0621       Set loop
FDC6   AC 22 06   LDY $0622       counter
FDC9³  50 FE      BVC $FDC9       Wait for Byte Ready
FDCB   B8         CLV             Reset Byte Ready Flag
FDCC   CA         DEX             Low-byte of counter
FDCD   D0 FA      BNE $FDC9       at null?
FDCF   88         DEY             YES—Then decrement Y
FDD0   10 F7      BPL $FDC9       Y times awaited 256 Byte Readys?
FDD2   60         RTS             YES—Return from this subroutine
-------------------------------------------------------------------
[FB09/FB59/FB7A/FBCB/FC12/FC2D/FD5F]
Stop control by format errors
FDD3   CE 20 06   DEC $0620       Number of format attempts -1
FDD6   F0 03      BEQ $FDDB       Run across a format error?
FDD8   4C 9C F9   JMP $F99C       NO—Then continue formatting
FDDB¹  A0 FF      LDY #$FF        Set 'Format to end'
FDDD   84 51      STY $51         flag
FDDF   C8         INY             Clear 'Buffer In GCR-Code'
FDE0   84 50      STY $50         flag
FDE2   4C 69 F9   JMP $F969       End formatting
-------------------------------------------------------------------
```

```
[FC98/FDEC]
Copy bytes in buffer 0 at 70 bytes over
(Y-register must contain the number of bytes to be copied)
FDE5  B9 00 03   LDA $0300,Y     Get byte from start of buffer and
FDE8  99 45 03   STA $0345,Y     transfer up
FDEB  88         DEY             Choose next byte
FDEC  D0 F7      BNE $FDE5       All bytes?
FDEE  AD 00 03   LDA $0300       YES—Then copy last
FDF1  8D 45 03   STA $0345       byte and
FDF4  60         RTS             return from this subroutine
-------------------------------------------------------------------
[FC9B]
Copy the range $01BB-$01FF in the buffer to which $30/$31 points
FDF5  A0 44      LDY #$44        Startposition $01FF
FDF7¹ B9 BB 01   LDA $01BB,Y     Get byte from interim buffer and
FDFA  91 30      STA ($30),Y     transfer to data buffer
FDFC  88         DEY             Choose next byte
FDFD  10 F8      BPL $FDF7       All bytes already transferred?
FDFF  60         RTS             YES—Return from this subroutine
-------------------------------------------------------------------
[8D59/9AE6/9CCC/FB1A/FD24/BF0C]
Switch head circuitry from write to read
FE00  AD 0C 1C   LDA $1C0C       Get control register and
FE03  09 E0      ORA #$E0        switch head to read
FE05  8D 0C 1C   STA $1C0C       (CB2 output =1)
FE08  A9 00      LDA #$00        Switch data port to head
FE0A  8D 03 1C   STA $1C03       for input
FE0D  60         RTS             Return from this subroutine
-------------------------------------------------------------------
[FCAE] Write $55 to entire track
FE0E  AD 0C 1C   LDA $1C0C       Get control register
FE11  29 1F      AND #$1F        and invert head for writing
FE13  09 C0      ORA #$C0        Bit 5-7 spread and set bit 6/7
FE15  8D 0C 1C   STA $1C0C       (CB2 output =0)
FE18  A9 FF      LDA #$FF        Switch head data port
FE1A  8D 03 1C   STA $1C03       to output
FE1D  A9 55      LDA #$55        Send $55 over
FE1F  8D 01 1C   STA $1C01       the write head
FE22  A2 28      LDX #$28        Set register counter to
FE24  A0 00      LDY #$00        write 10240 times
FE26¹ 50 FE      BVC $FE26       Electronics ready for next byte?
FE28  B8         CLV             YES—Reset flag again
FE29  88         DEY             Write 256 bytes
FE2A  D0 FA      BNE $FE26       256 Bytes already?
FE2C  CA         DEX             YES—Write 256 bytes 40 times
FE2D  D0 F7      BNE $FE26       40 writings completed?
FE2F  60         RTS             YES—Return from this subroutine
```

```
[9BEC/FC92]
Convert blockheader from binary into GCR
FE30  A9 00     LDA #$00        Reset buffer pointer
FE32  85 30     STA $30         to start
FE34  85 2E     STA $2E         Pointer low-byte to binary data
FE36  85 36     STA $36         Position in current buffer
FE38  A9 BB     LDA #$BB        Turn position pointer to
FE3A  85 34     STA $34         status buffer
FE3C  A5 31     LDA $31         Get pointer to current
FE3E  85 2F     STA $2F         data buffer
FE40  A9 01     LDA #$01        Set pointer to
FE42  85 31     STA $31         status buffer
FE44¹ A4 36     LDY $36         Determine current position
FE46  B1 2E     LDA ($2E),Y     Get byte from buffer and save
FE48  85 52     STA $52         as first byte to be converted
FE4A  C8        INY             Turn pointer to next byte
FE4B  B1 2E     LDA ($2E),Y     Get byte from buffer and save
FE4D  85 53     STA $53         as second byte to be converted
FE4F  C8        INY             Turn pointer to next byte
FE50  B1 2E     LDA ($2E),Y     Get byte from buffer and save
FE52  85 54     STA $54         as third byte to be converted
FE54  C8        INY             Turn pointer to next byte
FE55  B1 2E     LDA ($2E),Y     Get byte from buffer and save
FE57  85 55     STA $55         as third byte to be converted
FE59  C8        INY             Turn pointer to next byte
FE5A  F0 08     BEQ $FE64       Reached end of buffer?
FE5C  84 36     STY $36         NO—Save position
FE5E  20 D0 F6  JSR $F6D0       Cmpute 4binary bytes to 5GCRbytes
FE61  4C 44 FE  JMP $FE44       Continue conversion
FE64¹ 4C D0 F6  JMP $F6D0       Cmpute 4binary bytes to 5GCRbytes
------------------------------------------------------------------
[Originates at system vector FFFE]
FE67  6C A9 02  JMP ($02A9)     Jump to IRQ-Routine $9D88/$9DDE
------------------------------------------------------------------
FE6A  FF ...                    unused
FE84  ... FF                    ROM-area
------------------------------------------------------------------
Directory and BAM design
FE85  12                        Number of directory track(18)
FE86  04                        # of bytes for every track in BAM
FE87  04                        BAM start position in sector 18,0
FE88  90                        Beginning of disk name (Pos. 144)
------------------------------------------------------------------
```

```
Table of disk commands
FE89  56                        'V' :  Validate / Collect
FE8A  49                        'I' :  Initialize
FE8B  44                        'D' :  Duplicate (dual drives only)
FE8C  4D                        'M' :  Memory command
FE8D  42                        'B' :  Block command
FE8E  55                        'U' :  User command
FE8F  50                        'P' :  Position / Record
FE90  26                        '&' :  & - command
FE91  43                        'C' :  Copy
FE92  52                        'R' :  Rename
FE93  53                        'S' :  Scratch
FE94  4E                        'N' :  New / Header
-------------------------------------------------------------------
Addresses of disk commands
FE95  84 05 C1 F8 1B 5C 07 A3     Low-bytes of origin addresses
FE9D  F0 88 23 0D                 for the commands
FEA1  ED D0 C8 CA CC CB E2 E7     High-bytes of origin addresses
FEA9  C8 CA C8 EE                 for the commands
-------------------------------------------------------------------
Bit pattern for testing command syntax

Meaning of bits :
(1=been tested; corresponding bit in test value must be 0)

Bit0  '=' character on hand in command string
Bit1  Other parameters on hand after '=' character
Bit2  Several filenames for 2nd file designation
Bit3  Joker on hand in 2nd file designation
Bit6  Several filenames for 1st file designation
Bit7  Joker on hand in 1st file declaration

FEAD  51                          %01010001  Copy file(s)
FEAE  DD                          %11011101  Rename file
FEAF  1C                          %00011100  Scratch file(s)
FEB0  9E                          %10011110  Format diskette
FEB1  1C                          %00011100  Read file
-------------------------------------------------------------------
Identifier in command string for operating mode
FEB2  52 57 41 4D                 R, W, A, M
-------------------------------------------------------------------
File type identifier in command string
FEB6  44 53 50 55 4C              D, S, P, U, L
-------------------------------------------------------------------
```

```
Names of different file types
FEBB  44 53 50 55 52             1st char.:   D,  S,  P,  U,  R
FEC0  45 45 52 53 45             2nd char.:   E,  E,  R,  S,  E
FEC5  4C 51 47 52 4C             3rd char.:   L,  Q,  G,  R,  L
-----------------------------------------------------------------
Mask for LED-bit in control register
FECA  08 00                      Drive 0, drive 1 (not on hand)
-----------------------------------------------------------------
Set processor status flag
FECC  00                         N=0   V=0   Z=1
FECD  3F                         N=0   V=0   Z=0
FECE  7F                         N=0   V=1   Z=0
FECF  BF                         N=1   V=0   Z=0
FED0  FF                         N=1   V=1   Z=0
-----------------------------------------------------------------
Number of sectors in declared track range
FED1  11                         Track 31-35 :  17 sectors
FED2  12                         Track 25-30 :  18 sectors
FED3  13                         Track 18-24 :  19 sectors
FED4  15                         Track 01-17 :  21 sectors
-----------------------------------------------------------------
FED5  41                         Identifier for 1541-Format ('A')
-----------------------------------------------------------------
FED6  04                         Number of track changes
-----------------------------------------------------------------
Tracks that will be changed by the sector number and bitrate
FED7  24 1F 19 12                track nummbr 36, 31, 25 and 18
-----------------------------------------------------------------
FEDB  01 FF FF 01 00             Readerror ctrl bytes f/head-move
-----------------------------------------------------------------
Buffer position in memory
FEE0  03 04 05 06 07 07          High-bytes of buffer addresses
-----------------------------------------------------------------
FEE6  FF                         Empty byte (1541 DOS checksum)
-----------------------------------------------------------------
[FF0B]  Reset w/o hardware test; Pointer set through $EBC5
FEE7  6C 65 00   JMP ($0065)     Jump to $EB22
-----------------------------------------------------------------
[EA7A]
Initialize and switch LED
FEEA  8D 00 1C   STA $1C00       Set 'LED on' bit (8)
FEED  8D 02 1C   STA $1C02       and switch pin for output
FEF0  4C 7D EA   JMP $EA7D       Return to hardware error routine
-----------------------------------------------------------------
```

```
[BF36/E97D]
Bus delay for 1541 bus as opposed to 1540 bus
FEF3  8A         TXA              Retain X-register
FEF4  A2 05      LDX #$05         Set counter
FEF6* CA         DEX              42 cycles delay
FEF7  D0 FD      BNE $FEF6        Time up?
FEF9  AA         TAX              YES-Re-determine X-register
FEFA  60         RTS              Return from this subroutine
-----------------------------------------------------------------
[82AB/E980]
Output null bit
FEFB  20 AE E9   JSR $E9AE        Set clock output to high
FEFE  4C 9C E9   JMP $E99C        Set data output to low
-----------------------------------------------------------------
[Original at 'UI' command]
1541/1540 Bus mode switching
FF01  AD 02 02   LDA $0202        Get 3rd char. frm command string&
FF04  C9 2D      CMP #$2D         test with '-'
FF06  F0 05      BEQ $FF0D        Identical?
FF08  38         SEC              NO-Compare character
FF09  E9 2B      SBC #$2B         with '+'
FF0B  D0 DA      BNE $FEE7        Identical?
FF0D1 85 23      STA $23          YES-Set flag for bus mode
FF0F  60         RTS              Return from this subroutine
-----------------------------------------------------------------
[EAA4]
Input/Output initialization
FF10  8E 03 18   STX $1803        Set data direction for PA
FF13  A9 02      LDA #$02         [For error,see 7.1.5]
FF15  4C 5A A6   JMP $A65A        Continue
-----------------------------------------------------------------
[A664]
Set data direction for PB
FF18  A9 1A      LDA #$1A         %00011010
FF1A  8D 02 18   STA $1802        in data direction register
FF1D  4C A7 EA   JMP $EAA7        Back to reset
-----------------------------------------------------------------
[E9DC/FF25]
Data waits to equal low (phys. hih); set timer
FF20  AD 00 18   LDA $1800        Get bus control register and
FF23  29 01      AND #$01         test data line
FF25  D0 F9      BNE $FF20        Is data set?
FF27  A9 01      LDA #$01         NO-Start counter
FF29  8D 05 18   STA $1805        for 256 cycles
FF2C  4C DF E9   JMP $E9DF        Keep going
-----------------------------------------------------------------
```

```
[EE3D]
Format dislette
FF2F   A9 FF       LDA #$FF       Clear flag for current
FF31   85 51       STA $51        track
FF33   AD 0F 18    LDA $180F      Get control register
FF36   29 20       AND #$20       and test operating mode
FF38   D0 03       BNE $FF3D      Is drive in 1541 mode (1 MHz)?
FF3A   A9 24       LDA #$24       YES—Determine max. no. of tracks
FF3C   2C          .byte $2C      Jump to next 2 bytes(Bit command)
FF3D1  A9 47       LDA #$47       Number of tracks in 2-sided mode
FF3F   8D AC 02    STA $02AC      Set track number
FF42   4C 79 A7    JMP $A779      Format diskette
-----------------------------------------------------------------
[FA32]  cf. 87E7/9A66
One half-step outward
FF45   98          TYA            Retain
FF46   48          PHA            Y-register
FF47   A0 64       LDY #$64       # of pick-up attempts /tr.0 (100)
FF491  AD 0F 18    LDA $180F      Get control register A
FF4C   6A          ROR A          Put track0-ident. (bit0) in carry
FF4D   08          PHP            and save carry
FF4E   AD 0F 18    LDA $180F      Read control register again
FF51   6A          ROR A          Shift track0-ident. (bit0)
FF52   6A          ROR A          to bit7
FF53   28          PLP            Get previous pick-up result
FF54   29 80       AND #$80       Isolate last pick-up result
FF56   90 04       BCC $FF5C      Is track0 active in first test?
FF58   10 1D       BPL $FF77      NO—Has track0 now been reached?
FF5A   30 02       BMI $FF5E      YES—Jump to $FE5E
FF5C1  30 19       BMI $FF77      Is track 0 still active?
FF5E1  88          DEY            YES—Try again
FF5F   D0 E8       BNE $FF49      All tries executed?
FF61   B0 14       BCS $FF77      YES—Is head at track0-position?
FF63   AD 00 1C    LDA $1C00      YES—Cntrl register for step-motor
FF66   29 03       AND #$03       Isolate stepper bits
FF68   D0 0D       BNE $FF77      Is a stepper coil under control?
FF6A   A5 7B       LDA $7B        NO—Set head cntrl byte/read error
FF6C   D0 09       BNE $FF77      Head in position?
FF6E   68          PLA            NO—Re-establish
FF6F   A8          TAY            Y-register
FF70   A9 00       LDA #$00       Clear number of steps done by
FF72   85 4A       STA $4A        stepper
```

```
FF74   4C BE FA   JMP $FABE        Initialize head
FF77⁴  68         PLA              Re-establish
FF78   A8         TAY              Y-register
FF79   E6 4A      INC $4A          Move another step out
FF7B   AE 00 1C   LDX $1C00        Get control register and set
FF7E   CA         DEX              head to move one
FF7F   4C 38 FA   JMP $FA38        step outward
-------------------------------------------------------------------

[903D/EBC2]
Initialize 1541 mode
FF82   20 59 F2   JSR $F259        Disk controller reset
FF85   A9 05      LDA #$05         Determine IBM-34
FF87   85 3C      STA $3C          sector layout
FF89   A9 88      LDA #$88         Turn IRQ vectors
FF8B   8D A9 02   STA $02A9        to routine
FF8E   A9 9D      LDA #$9D         $9D88 (1541
FF90   8D AA 02   STA $02AA        interrupt)
FF93   A9 24      LDA #$24         Set maximum number
FF95   8D AC 02   STA $02AC        of tracks 35)
FF98   18         CLC              Flag for 'side 1'
FF99   4C F3 93   JMP $93F3        Choose head
-------------------------------------------------------------------

[EE1D]
Activate drive
FF9C   85 FF      STA $FF          Set drive status
FF9E   4C 00 C1   JMP $C100        LED on
-------------------------------------------------------------------

[D610]
Set head control byte
FFA1   85 7B      STA $7B          Set byte in pointer
FFA3   4C 76 D6   JMP $D676        Go back
-------------------------------------------------------------------

[D628]
Rest positioning mode to next track
FFA6   20 76 D6   JSR $D676        Control head
FFA9   A9 00      LDA #$00         Clear 'head mode'
FFAB   85 7B      STA $7B          flag
FFAD   60         RTS              Return from this subroutine
-------------------------------------------------------------------

[CD91]
Set buffer pointer to 'B-W'
FFAE   A4 82      LDY $82          Get channel number
FFB0   4C DE D3   JMP $D3DE        Set pointer
-------------------------------------------------------------------

FFB3   FF ...                      Unused
FFE5   ... FF                      ROM-area
-------------------------------------------------------------------
```

```
[Not used in 1571 DOS]
DOS system vectors
FFE6  C6 C8                   Format diskette
$C8C6
FFE8  8F F9                   Switch off drive motor
$F98F
-----------------------------------------------------------
User vectors; jump addresses of User commands
FFEA  5F CD              U1 or UA  : Read sector         $CD5F
FFEC  CD 97              U2 or UB  : Write sector        $CD97
FFFE  03 05              U3 or UC  : Jump to buffer 2    $0500
FFF0  06 05              U4 or UD  : Jump to buffer 2    $0503
FFF2  09 05              U5 or UE  : Jump to buffer 2    $0506
FFF4  0C 05              U6 or UF  : Jump to buffer 2    $0509
FFF6  05 0F              U7 or UG  : Jump to buffer 2    $050C
FFF8  0F 05              U8 or UH  : Jump to buffer 2    $050F
FFFA  01 FF              U9 or UI  : Toggle  1540/41     $FF01
-----------------------------------------------------------
System vectors
FFFC  A0 EA              U: or UJ  : Execute reset       $EAA0
FFFE  67 FE              IRQ vector  (Bus/Disk controller) $FE67
-----------------------------------------------------------
```

Appendix B

The 1570 DOS (1571 Revisions)

The Commodore 1570 disk drive is a single-sided drive that utilizes 1571 electronics, and is currently available only in Europe. Because this book is marketed internationally, and as Commodore may release the the 1570 in the United States, we have included this section detailing the differences between the two drives.

The 1570 disk drive has almost the same operating system as the 1571 drive, and so is treated as a modified 1571 ROM. In fact, the hardware of the two drives is almost identical.

The biggest difference is that the 1570 drive is a single-sided drive (i.e., no two-sided read/write heads). All ROMs have been modified accordingly, making this drive operate with the same BAM as a 1541 drive.

There are changes in the 1570's two motors. The stepper motor is not as efficient as that of the 1571 drive, and the 1570 motors are simply not as fast as those of the 1571. Thus, time constants for motor control have been changed.

A few small errors have been cleared up in the 1570 which existed in the 1571 series.

1570 DOS

```
------------------------------------------------------------------
8000  75 98                      1570 ROM checksum
------------------------------------------------------------------
84E4  20 4D AA   JSR $AA4D       Format diskette
------------------------------------------------------------------
8827  A0 08      LDY #$08        1541 stepper motor delay
------------------------------------------------------------------
8FD4  4C 21 90   JMP $9021       Display '31 Syntax Error'
------------------------------------------------------------------
90D9  20 5B AA   JSR $AA5B       Test filetype for 'PRG'
------------------------------------------------------------------
A40F  8D D7 FE   STA $FED7       Set highest track number
------------------------------------------------------------------
A445  60         RTS             Separate from
A446  EA         NOP             2nd side of diskette
------------------------------------------------------------------
```

```
A4DD  53 54 45 56 45 20 4C 41  Copyright for 1570 modification
A4E5  4D 0D                    by Steve Lam
-------------------------------------------------------------------
A597  AD D7 FE   LDA $FED7     Get highest track number
-------------------------------------------------------------------
A671  AD D7 FE   LDA $FED7     Get highest track number
-------------------------------------------------------------------
A6C4  8D D7 FE   STA $FED7     Set highest track number
-------------------------------------------------------------------
A6DF  8D D7 FE   STA $FED7     Set highest track number
-------------------------------------------------------------------
A726  8D D7 FE   STA $FED7     Set highest track number
-------------------------------------------------------------------
A773  8D D7 FE   STA $FED7     Set highest track number
-------------------------------------------------------------------
A7B3  20 62 AA   JSR $AA62     Get control register
-------------------------------------------------------------------
A7C7  4C CE A7   JMP $A7CE     Don't test 1541 mode
-------------------------------------------------------------------
A7D7  AD D7 FE   LDA $FED7     Get highest track number
-------------------------------------------------------------------
A941  AD D7 FE   LDA $FED7     Get highest track number
-------------------------------------------------------------------
[D867] Test for error acknowledgements
AA3F  C9 02      CMP #$02      Compare with first error number
AA41  90 07      BCC $AA4A     Is there an error?
AA43  C9 0F      CMP #$0F      NO—Test for 'Drive not ready'
AA45  F0 03      BEQ $AA4A     Is drive ready?
AA47  4C 6B D3   JMP $D36B     YES—Return to main routine
AA4A1 4C 73 D3   JMP $D373     Observe error
-------------------------------------------------------------------
[84E4]  Format diskette
AA4D  85 51      STA $51       Set current track to be formatted
AA4F  20 7C 87   JSR $877C     Drive LED on
AA52  20 89 A9   JSR $A989     Format diskette
AA55  48         PHA           Retain acknowledgement
AA56  20 88 87   JSR $8788     Drive LED off
AA59  68         PLA           Repeat acknowledgement
AA5A  60         RTS           Return from this subroutine
-------------------------------------------------------------------
[90D9]  Test filetype for program file
AA5B  A5 E7      LDA $E7       Get filetype byte
AA5D  29 07      AND #$07      and isolate filetype flags
AA5F  C9 02      CMP #$02      Compare with 'PRG'
AA61  60         RTS           Return from this subroutine
-------------------------------------------------------------------
```

```
[AA62] Read control register
AA62  AD 0F 18   LDA #$180F    Get value of control register
AA65  2C 01 18   BIT #$1801    Read in new value in control
register
AA68  60         RTS           Return from this subroutine
-----------------------------------------------------------------
CD22  CD D7 FE   CMP $FED7     Compare with maximum track
-----------------------------------------------------------------
D03F  4C 8C D5   JMP $D58C     Execute job
-----------------------------------------------------------------
D05D  20 86 D5   JSR $D586     Read BAM from diskette
-----------------------------------------------------------------
D097  9D FA 02   STA $02FA,X   High-byte of number of free blocks
-----------------------------------------------------------------
D367  4C 3F AA   JMP $AA3F     Test acknowledgement
-----------------------------------------------------------------
D51D  CD D7 FE   CMP $FED7     Compare with maximum track
-----------------------------------------------------------------
D563  CD D7 FE   CMP $FED7     Compare with maximum track
-----------------------------------------------------------------
E5C7  80                       Last character of on-message
-----------------------------------------------------------------
ED8F  20 B7 EE   JSR $EEB7     Create new BAM
-----------------------------------------------------------------
EE40  20 05 F0   JSR $F005     Set buffer for BAM
-----------------------------------------------------------------
EEB1  20 60 D4   JSR $D460     Read sector
-----------------------------------------------------------------
EF28  20 20 F2   JSR $F220     Test number of blocks free
-----------------------------------------------------------------
EF2F  CD D7 EE   CMP $FED7     Compare with maximum track
-----------------------------------------------------------------
EF37  4C 8A D5   JMP $D58A     Write BAM to diskette
-----------------------------------------------------------------
EF5F  20 CF EF   JSR $EFCF     Set buffer pointer
-----------------------------------------------------------------
EF93  20 CF EF   JSR $EFCF     Set buffer pointer
-----------------------------------------------------------------
F001  4C 8A D5   JMP $D58A     Write BAM to diskette
-----------------------------------------------------------------
F005  20 3A EF   JSR $EF3A     Set buffer pointer
-----------------------------------------------------------------
F09C  4C 8A D5   JMP $D58A     Write BAM to diskette
-----------------------------------------------------------------
F107  4C 86 D5   JMP $D586     Read BAM from diskette
-----------------------------------------------------------------
```

```
F12D  A5 6F     LDA $6F      Reserve current
F12F  48        PHA          BAM pointer
-------------------------------------------------------------------
F147  CD D7 EE  CMP $FED7    Compare with maximum track
-------------------------------------------------------------------
F1C4  20 11 F0  JSR $F011    Set BAM pointer
-------------------------------------------------------------------
F1D5  CD D7 EE  CMP $FED7    Compare with maximum track
-------------------------------------------------------------------
F1DF  20 11 F0  JSR $F011    Set BAM pointer
-------------------------------------------------------------------
F24B  AE D6 FE  LDX $FED6    Number of track zones on diskette
-------------------------------------------------------------------
F98A  A9 7D     LDA #$7D     'Motor on' delay about 1/0.5 sec.
-------------------------------------------------------------------
FF3F  8D D7 FE  STA $FED7     Mark maximum track number
-------------------------------------------------------------------
FF95  8D D7 FE  STA $FED7     Mark maximum track number
-------------------------------------------------------------------
```

Appendix C

1571 Zeropage Listing

```
-------------------------------------------------------------------
     0  -  5  Jobcode of corresponding buffer assignment (0-5)
    $0  - $5  Buffer 5 is not allocated in RAM.

              Meanings of jobcodes:
              $80 Read a sector
              $88 Read sector from same track
              $90 Write a sector
              $A0 Verify a sector
              $B0 Look for a sector header
              $C0 Set head to track 0
              $D0 Execute program in buffer
              $E0 Combine program in jobloop
              $F0 Format diskette

              Meanings of acknowledgements:

              $00/01 No errors
              $02 Blockheader not found
              $03 Sync-mark not found
              $04 Data block not found
              $05 Data block checksum wrong
              $06 Format error
              $07 Verify error
              $08 Write-protect on hand
              $09 Wrong header checksum
              $0A Data block too long
              $0B False ID / diskette changed
              $0D Index hole not found
              $0E CP/M syntax error
              $0F No disk found

-------------------------------------------------------------------
     6 -  17 Respective track/sector number for buffers 0-5
    $6 - $11 e.g., 6 contains the track and 7 the sector for
             buffer 0
-------------------------------------------------------------------
    18 -  19 First and second characters of disk ID
   $12 - $13 in drive 0
-------------------------------------------------------------------
    20 -  21 Unused
   $14 - $15 memory
-------------------------------------------------------------------
```

```
  22 -  23 First and second ID characters of last-read
 $16 - $17 sector header
-----------------------------------------------------------------
  24 -  25 Track and sector number of
 $18 - $19 last-read sector header
-----------------------------------------------------------------
  26       Checksum of last-read
 $1A       sector header
-----------------------------------------------------------------
  27       Control byte of routine at $86E6
 $1B       Buffer pointer on format routine $9B89
-----------------------------------------------------------------
  28       'Diskette initialized' flag
 $1C       0=no  <>=yes
-----------------------------------------------------------------
  29       Like 28/$1C, but for drive 1
 $1D       Value always 1 [EBBA]
-----------------------------------------------------------------
  30       Current status of write-protect notch
 $1E       0=write protect active; 1=no write protect
-----------------------------------------------------------------
  31  $1F  Unused
-----------------------------------------------------------------
  32       Operating status of drive 0
 $20       Bit 4: 1=motor runs until turned off
           Bit 5: 1=motor switched on
           Bit 6: 1=stepper motor active, head set
           Bit 7: 1=drive not ready
-----------------------------------------------------------------
  33  $21  Like 32/$20, only for drive 1
-----------------------------------------------------------------
  34  $22  Track number of current job
-----------------------------------------------------------------
  35  $23  Flag for bus mode: 0=1541 bus  <>0= 1540 bus
-----------------------------------------------------------------
  36 - 43  Sector header buffer
 $24 - $2B Commodore sectors : Data in GCR-code
           CP/M sectors      : ID array contents
-----------------------------------------------------------------
  44 -  45 Unused
 $2C - $2D memory
-----------------------------------------------------------------
  46 -  47 Pointer to current buffer position
 $2E - $2F converted to GCR
-----------------------------------------------------------------
```

```
  48 -  49 Pointer to position in
 $30 - $31 current data buffer
-------------------------------------------------------------------
  50 -  51 Pointer to track/sector of jobloop (6-17)
 $32 - $33 by formatting: pointer to current sector header
-------------------------------------------------------------------
  52   $34  Pointer converted to current GCR byte
-------------------------------------------------------------------
  53   $35  Jobloops yet to be run by 'motor out'
-------------------------------------------------------------------
  54   $36  Pointer to position of binary byte by GCR conversion
-------------------------------------------------------------------
  55   $37  Bus status byte:
            Bit 0 1=Flag for 'file only has one sector'
            Bit 3 Reverse status of Clock line
            next, waiting for Clock signal
            Bit 6 1=1571 bus mode 0=1541 bus mode
            Bit 7 1=1571 operating mode (2mHz)
                  0=1541 operating mode (1mHz)
-------------------------------------------------------------------
  56   $38  Identifer for last-read header (normal 7)
-------------------------------------------------------------------
  57   $39  Data block identifier (8)
-------------------------------------------------------------------
  58   $3A  Checksum of buffer/data block
-------------------------------------------------------------------
  59   $3B  'U0' command number [see $8030]
-------------------------------------------------------------------
  60   $3C  IBM system 34 sector format (after reset 5)
-------------------------------------------------------------------
  61   $3D  Drive number of current job
-------------------------------------------------------------------
  62   $3E  Number of active drives ($ff=no drives)
-------------------------------------------------------------------
  63   $3F  Buffer number of current job
-------------------------------------------------------------------
  64   $40  Track of last job
-------------------------------------------------------------------
  65   $41  Number of last job [used only in F340/F44B]
-------------------------------------------------------------------
  66   $42  Difference between new and old track
-------------------------------------------------------------------
  67   $43  Number of sectors per track
-------------------------------------------------------------------
  68   $44  Number of CP/M sub-sectors per sector
-------------------------------------------------------------------
```

```
    69  $45  Jobcode command bits (bits 3-6 of original jobcode)
------------------------------------------------------------------
    70  $46  Next character to be sent over 1571 bus
------------------------------------------------------------------
    71  $47  Data block identifier (80
------------------------------------------------------------------
    72  $48  Runtime counter for motor
------------------------------------------------------------------
    73  $49  Temporary storage for stack pointer
------------------------------------------------------------------
    74  $4A  Bits 0-6 : Number of half-track-step travel
             Bit 7    : 0=step out
                      : 1=step in
------------------------------------------------------------------
    75  $4B  Assorted temporary storage
    76  $4C  Sector difference to next optimal job
------------------------------------------------------------------
    77  $4D  Sector number of next optimal job
------------------------------------------------------------------
    78 -  79 Temporary storage of current buffer pointer
   $4E - $4F from GCR conversion
------------------------------------------------------------------
    80  $50  Buffer data format flag
             0=binary  <>=GCR
------------------------------------------------------------------
    81  $51  Current track of format
             $FF=Format not in process
------------------------------------------------------------------
    82 -  85 Temporary storage for 4 binary bytes, which will
   $52 - $55 be converted to 5 GCR bytes
------------------------------------------------------------------
    86 -  93 Temporary storage for 8 GCR values, to produce 8
             binary half-bytes,
   $56 - $5D and from that 4 binary bytes
------------------------------------------------------------------
    94  $5E  Command status byte
             Bit 0-4: Last CP/M error message in jobloop
             Bit 7:  1=Disk in IBM System 34 format
------------------------------------------------------------------
    95  $5F  Current jobcode
------------------------------------------------------------------
    96  $60  IBM-34 format: Smallest sector number per track
------------------------------------------------------------------
    97  $61  IBM-34 format: Largest sector number per track
------------------------------------------------------------------
```

```
 98 -  99 1541 mode: Pointer to current head control routine
$62 - $63 1571 mode: Pointer to positioning phase
-------------------------------------------------------------------
100   $64 Current head position in half-track steps
-------------------------------------------------------------------
101 - 102 Pointer to reset (no hardware test) $EB22
$65 - $66 will jump from $FEE7 when 'UI' has not '+' or '-'
-------------------------------------------------------------------
103   $67 Target track
-------------------------------------------------------------------
104   $68 Flag for initializer method (always 0) [set by C63D]
          0=Automatic initialization
          <>0=Initialized by 'hand' (i-command)
-------------------------------------------------------------------
105   $69 Sector format for Commodore diskettes (6)
-------------------------------------------------------------------
106   $6A Bits 0-5: Number of 'bad' read attempts
          Bit 6   : Head not set next to track
          Bit 7   : Track 0 not run
-------------------------------------------------------------------
107 - 110 Pointer to table of 1541 User-command
$6B - $6C ($FFEa)
-------------------------------------------------------------------
111 - 114 Temporary storage for sundries
$6F - $72 (BAM calculations, etc.)
-------------------------------------------------------------------
115   $73 Number of side-sectors to relative file
-------------------------------------------------------------------
116   $74 Unused memory
-------------------------------------------------------------------
117 - 118 Address pointer for different
$75 - $76 system operations
-------------------------------------------------------------------

119   $77 Device address for Listen + 20(flag in command byte)
-------------------------------------------------------------------
120   $78 Device address for Talk + 20 (flag in command byte)
-------------------------------------------------------------------
121   $79 Listen flag (1=listener mode)
-------------------------------------------------------------------
122   $7A Talk flag (1=talk mode)
-------------------------------------------------------------------
123   $7B Current head positioning control byte from readerror
-------------------------------------------------------------------
124   $7C 'ATN encountered' flag
-------------------------------------------------------------------
```

```
125  $7D  'ATN observed' flag 0=yes; <>0= ATN ignored
--------------------------------------------------------------------
126  $7E  Track number of last access
--------------------------------------------------------------------
127  $7F  Current drive number
--------------------------------------------------------------------
128  $80  Current track number
--------------------------------------------------------------------
129  $81  Current sector number
--------------------------------------------------------------------
130  $82  Current internal channel number (0-6)
--------------------------------------------------------------------
131  $83  Current secondary address
--------------------------------------------------------------------
132  $84  Last command word sent over serial bus
--------------------------------------------------------------------
133  $85  Current 1541 bus data byte
--------------------------------------------------------------------
134 -138  Temporary storage for
$86 -$8A  assorted purposes
--------------------------------------------------------------------
139 -142  Math register 1
$8B -$8E
--------------------------------------------------------------------
142 -147  Math register 2
$8F -$93
--------------------------------------------------------------------
148 -149  Pointer in directory buffer
$94 -$95
--------------------------------------------------------------------
150  $96  Number of first System 34 sector read
--------------------------------------------------------------------
151  $97  Number of System 34 sectors per track
--------------------------------------------------------------------
152  $98  Bit counter for bits per byte (for data transfer)
--------------------------------------------------------------------
153 -154  Pointer to start of
$99 -$9A  buffer 0 ($0300)
--------------------------------------------------------------------
155 -156  Pointer to start of
$9B -$9C  buffer 1 ($0400)
--------------------------------------------------------------------
157 -158  Pointer to start of
$9D -$9E  buffer 2 ($0500)
--------------------------------------------------------------------
```

```
159 - 160 Pointer to start of
$9F - $A0 buffer 3 ($0600)
---------------------------------------------------------------------
161 - 162 Pointer to start of
$A1 - $A2 buffer 4 ($0700)
---------------------------------------------------------------------
163 - 164 Pointer to start of
$A3 - $A4 input buffer ($0200)
---------------------------------------------------------------------
165 - 166 Pointer to start of
$A5 - $A6 error message buffer ($02D5)
---------------------------------------------------------------------
167 - 173 Channel buffer table 1:
$A7 - $AD Arranged one of the first buffers to internal
          channels

          167-173 correspond to channels 0-6

          Meaning of bytes:

          Bits 0-5: Buffer number arranged in channel
          Bit 6   : 1=Rewrite buffer contents
          Bit 7   : 0=Buffer used active
          $FF     : No buffer separated
---------------------------------------------------------------------
174 - 180 Channel buffer table 2:
$AE - $B4 arrange 2nd buffer (functions like 167-173 above)
---------------------------------------------------------------------
181 - 186 Number of blocks allocated to file by internal
          channel
$B5 - $BA (low-byte) Index: Channel number $82
---------------------------------------------------------------------
187 - 192 Number of blocks allocated to file by internal
          channel
$BB - $C0 (high-byte) Index: Channel number $82
---------------------------------------------------------------------
193 - 198 Pointer to current databyte of file by internal
          channel
$C1 - $C6 Index: Channel number $82
---------------------------------------------------------------------
199 - 204 Record length of relative file opened via
$C7 - $CC internal channel
          Index: Channel number $82
---------------------------------------------------------------------
205 - 210 Channel buffer table 3:
$CD - $D2 Organize 3rd buffer (see 167-173)
---------------------------------------------------------------------
```

```
     211   $D3  Pointer to first filename
---------------------------------------------------------------------
     212   $D4  Position in current record
---------------------------------------------------------------------
     213   $D5  Side-sector number
---------------------------------------------------------------------
     214   $D6  Pointer to record in side-sector
---------------------------------------------------------------------
     215   $D7  Pointer to data set of relative file
---------------------------------------------------------------------
     216 - 220 Directory filename table
     $D8 - $DC Directory sector where filename is found
---------------------------------------------------------------------
     221 - 225 Filename position table
     $DD - $E1 marks diectory entry area
---------------------------------------------------------------------
     226 - 230 Filename-specified drive table
     $E2 - $E6
---------------------------------------------------------------------
     231 - 235 Filename/filetype table
     $E7 - $EB
---------------------------------------------------------------------
     236 - 241 Channel number/filetype table
     $EC - $F1 Bit 0   : Drive number (0/1)
               Bits 1-3:Filetype
---------------------------------------------------------------------
     242 - 247 Channel number status table
     $F2 - $F7 Bit 1   :1=channel is write channel
               Bit 3   :0=EOF flag set
               Bit 7   :1=channel is write channel
---------------------------------------------------------------------
     248   $F8  EOI flag (last char.); 0=YES 1=NO
---------------------------------------------------------------------
     249   $F9  Current buffer number
---------------------------------------------------------------------
     250 - 254 Table for buffer-contained channel number
     $FA -$FE
---------------------------------------------------------------------
     255    $FF Drive status (drive 0): 0=drive ready
---------------------------------------------------------------------
     256   $100 Drive status (drive 1): 0=drive ready
---------------------------------------------------------------------
     257 - 325 Hardware stack of
     $101-$145 processor
---------------------------------------------------------------------
```

```
326 - 431 BAM buffer 2 for 1571 diskettes
$146-$1AF
----------------------------------------------------------------------
432  $1B0 CP/M error message
----------------------------------------------------------------------
433  $1B1 Flag for current diskette side: 0=side 1
----------------------------------------------------------------------
443 - 511 Status buffer to take GCR data
$1BB-$1FF
----------------------------------------------------------------------
512 - 553 Input buffer for command strings
$200-$229 from computer
----------------------------------------------------------------------
554  $22A Current command number; $FF=no command
----------------------------------------------------------------------
555 - 573 Secondary address table-internal channel
$22B-$23D Bits 0-3 : internal channel number
          Bit 6    : 1=channel for reading
          Bit 7    : 1=write channel
          $FF      : no secondary address
----------------------------------------------------------------------
574 - 579 Channel number table - current data byte
$23E-$243
----------------------------------------------------------------------
580 - 585 Channel number table - # of bytes to be transferred
$244-$249
----------------------------------------------------------------------
586  $24A Current filetype
----------------------------------------------------------------------
587  $24B Length of current filename in command string
----------------------------------------------------------------------
588  $24C Temporary storage for OPEN secondary address
----------------------------------------------------------------------
589  $24D Combine with call from D506 in jobcode
----------------------------------------------------------------------
590  $24E Max. number of sectors in current track
----------------------------------------------------------------------
591 - 592 Buffer assignment table
$24F-$250 Every bit of 16-bit value represents a buffer
          1=buffer assigned; 0=buffer free
----------------------------------------------------------------------
593  $251 'Newly written BAM, illegal'; 1=yes 0=no
----------------------------------------------------------------------
594  $252 As above, for drive 1
----------------------------------------------------------------------
595  $253 Flag for 'File found'; $FF=no
----------------------------------------------------------------------
```

```
     596  $254 'Directory in buffer' flag; 0=yes <>0=no
--------------------------------------------------------------------
     597  $255 Command mode
--------------------------------------------------------------------
     598  $256 Bitmap for channel assignment
               1=channel free; 0=channel used
--------------------------------------------------------------------
     599  $257 Pointer to current active buffers from 2-buffer
               operation
--------------------------------------------------------------------
     600  $258 Record length
--------------------------------------------------------------------
     601  $259 Current side-sector (track)
--------------------------------------------------------------------
     602  $25A Current side-sector
--------------------------------------------------------------------
     603 - 607 Table for buffer-jobcode
     $25B-$25F last jobcode of buffer
--------------------------------------------------------------------
     608 - 613 Table for channel - data sector (track number)
     $260-$265
--------------------------------------------------------------------
     614 - 619 Table for channel - data sector (sector number)
     $266-$26B
--------------------------------------------------------------------
     620  $26C Error number/blink counter
--------------------------------------------------------------------
     621  $26D LED mask from error blinking
--------------------------------------------------------------------
     622  $26E Last active drive [D7D1/D9FE]
--------------------------------------------------------------------
     623  $26F Last sector number [D7DC/DA03]
--------------------------------------------------------------------
     624  $270 Current channel number
--------------------------------------------------------------------
     625  $271 Number of bytes per IBM sub-sector
--------------------------------------------------------------------
     626 - 627 Temporary storage of directory entry
     $272-$273 (e.g., for block amount, etc.)
--------------------------------------------------------------------
     628  $274 Length of command string in input buffer
--------------------------------------------------------------------
     629  $275 Characters to be sought in input buffer
               [C165/C16D/C268/C273]
--------------------------------------------------------------------
     630  $276 Length of current filename in input buffer
--------------------------------------------------------------------
```

```
     631   $277 Number of filenames for 1st file declaration
-----------------------------------------------------------------
     632 - 639 Filename position table in input buffer points
     $27A-$27F to beginning of command string
-----------------------------------------------------------------
     640 - 644 Filename track table to current sector
     $280-$284
-----------------------------------------------------------------
     645 - 649 Filename number table of current sector
     $285-$289
-----------------------------------------------------------------
     650   $28A Joker flag; 0=no joker
-----------------------------------------------------------------
     651   $28B Command syntax byte
-----------------------------------------------------------------
     652   $28C Number of drives to be accessed (0/1/2)
-----------------------------------------------------------------
     653   $28D Flag for directory from both drives; 0=no
-----------------------------------------------------------------
     654   $28E Number of last drive
-----------------------------------------------------------------
     655   $28F Position of current directory entry
-----------------------------------------------------------------
     656   $290 Sector of current file entry
-----------------------------------------------------------------
     657   $291 Sector of current file entry
-----------------------------------------------------------------
     658   $292 Pointer to valid entry
-----------------------------------------------------------------
     659   $293 Pointer to next directory sector
-----------------------------------------------------------------
     660   $294 Position in directory sector
-----------------------------------------------------------------
     661   $295 Counter for directory entries per sector (8)
     662   $296 Filetype from command string; 0=no assignment
-----------------------------------------------------------------
     663   $297 File operation mode 0/1=read/write 2=append 3=modify
-----------------------------------------------------------------
     664   $298 'Error from job observed' flag; >128=no <128=yes
-----------------------------------------------------------------
     665   $299 Pointer to position phase from read error
-----------------------------------------------------------------
     666   $29A Control byte for head positioning by read error
-----------------------------------------------------------------
     667 - 668 Pointer to current BAM-track storage
     $29B-$29C for drive 0 and 1
-----------------------------------------------------------------
```

```
669 - 670  Track number assigned by the
$29D-$29E  BAm temporary storage
----------------------------------------------------------------
673 - 680  BAM temporary storage
$2A1-$2A8
----------------------------------------------------------------
681 - 682  IRQ vector from FE67
$2A9-$2AA
----------------------------------------------------------------
683   $2AB Assign motor runtime counter from diskette
----------------------------------------------------------------
684   $2AC Number of greates track+1 of diskette
----------------------------------------------------------------
685 - 686 Pointer in BAM buffer
$2AD-$2AE (temporary storage reserved at pointer)
----------------------------------------------------------------
687   $2AF '1541/1571 IRQ toggle' flag; 1=no
----------------------------------------------------------------
688 - 715 Produce buffer at
$2B0-$2CB directory line
----------------------------------------------------------------
716 - 724 Unused
$2CC-$2D4 memory
----------------------------------------------------------------
725 - 760 Generate buffer for error text
$2D5-$2F8 message
----------------------------------------------------------------
761   $2F9 'Invalid BAM' flag; 0=no 1=yes
----------------------------------------------------------------
762 - 763 Number of blocks free in drives
$2FA-$2FB 0 and 1 (low-bytes)
----------------------------------------------------------------
764 - 765 Number of blocks free in drives
$2FC-$2FD 0 and 1 (high-bytes)
----------------------------------------------------------------
766 - 767 Control byte for positioning next track for
$2FE-$2FF drives 0 and 1
----------------------------------------------------------------
```

Appendix D

Overview of Disk Errors

```
NUMBER  DEFINITION
------------------------------------------------------------------
00      OK

01      FILES SCRATCHED,XX
        Acknowledgement of scratch
        XX gives number of files deleted
------------------------------------------------------------------
TT=track; SS=sector at which error occurred
------------------------------------------------------------------
20      READ ERROR,TT,SS
        Sector header of a block was not found. The disk is treated
        as unformatted or bad.

21      READ ERROR,TT,SS
        Sync marker not found. Either disk is unformatted or there
        is a drive error, such as a misaligned read head, etc.

22      READ ERROR,TT,SS
        Data block of a sector has not been found.

23      READ ERROR,TT,SS
        Checksum error. When this happens, you will have to look
        into the sector several times with direct access commands,
        until the error is found. Then, you will have to read the
        sector into the disk buffer, and rewrite the sector. This
        re-computes the checksum, although the contents of the
        sector can be incorrect.

24      READ ERROR,TT,SS
        Error caused by hardware trouble—invalid bit pattern.

25      WRITE ERROR,TT,SS
        Writing a sector has caused a discrepancy determined by a
        verify error. Use a new diskette.

26      WRITE PROTECT ON,TT,SS
        The diskette is guarded by a write-protect tab.

27      READ ERROR,TT,SS
        Checksum error detected in sector header.
```

29	DISK ID MISMATCH,TT,SS Sector header ID doesn't match with last-read ID. Cause: Initialized or newly-formatted disk.
30	SYNTAX ERROR The 1571/1570 does not recognize the command sent over the command channel.
31	SYNTAX ERROR Command cannot be executed.
32	SYNTAX ERROR Command sent over channel is longer than 41 characters, and input buffer is full.
33	SYNTAX ERROR Joker has been used by writing as filename.
34	SYNTAX ERROR Filename was not found. Eventually, the characters after the command colon were forgotten.
39	FILE NOT FOUND Autoboot file given not found.
50	RECORD NOT PRESENT Data set of a relative file does not exist. This message can be ignored when first writing a data set, only to have it show itself when trying to read that file.
51	OVERFLOW IN RECORD Data being transferred to the disk is larger than the data set, so any more characters are ignored.
52	FILE TOO LARGE Number of last data set is too large; no more files can be fit onto the diskette.
60	WRITE FILE OPEN An attempt is made to access a file not closed by the normal methods. This file can only be re-opened using 'modify'.
61	FILE NOT OPEN An un-opened file is sought.

62 FILE NOT FOUND
Given program or file is not found.

63 FILE EXISTS
The new file already exists on diskette.

64 FILE TYPE MISMATCH
The given filetype doesn't match any filetype given on disk.

65 NO BLOCK,TT,SS
The block given by Block-Allocate is already occupied. TT and SS give the track and sector of the next free block of the track. If TT and SS=0, there are no more free sectors. See Chapter 2.1.3 for Block-Allocate and error handling for that command.

66 ILLEGAL TRACK OR SECTOR,TT,SS
The sector parameters given by direct access commands are wrong.

67 ILLEGAL TRACK OR SECTOR,TT,SS
The sector linking points to a sector which is not onhand.

70 NO CHANNEL
No more channels are available. You will have to close an already-open file somewhere to get a channel back.

71 DIR ERROR,TT,SS
The BAM contents in disk memory do not match with the BAM on diskette. You will have to initialize the diskette when this happens.

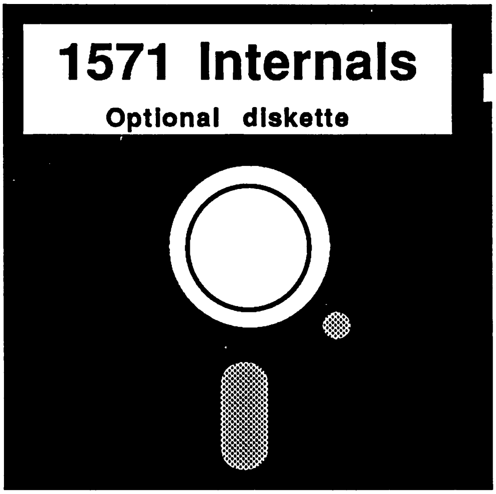
72 DISK FULL
You have reached the maximum capacity of the disk, and have less than three blocks free.

73 Power-on message
An attempt has been made to write to a disk formatted under another DOS.

74 DRIVE NOT READY
There is no formatted disk in the drive.

Index

Optional Diskette



Includes <u>Disk Monitor</u>!

For your convenience, the program listings contained in this book are available on a Commodore formatted floppy disk. You should order the diskette if you want to use the programs, but don't want to type them in from the listings in the book.

All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for $14.95 plus $2.00 ($5.00 foreign) for postage and handling.

When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

Abacus Software
P.O. Box 7219
Grand Rapids, MI 49510

Or for *fast* service, call **(616) 241-5510.**

COMMODORE® 1571™

The essential reference for all 1571 users

INTERNALS

For the beginner to the advanced user, **1571 Internals** is a vital addition to your C-128 computer library. Packed with straight-forward, detailed information on how to get the most out of your 1571 drive. Just a few of the topics covered in **1571 Internals**:

- Fundamentals for beginners
- Applying the disk drive commands
- Creating sequential and relative files
- Using your 1571 under Commodore BASIC
- Working with "foreign" disk formats
- Putting programs in the DOS buffer
- The 1571 and CP/M™
- Internal disk drive functions
- Fully documented 1571 DOS listing

About the author:
Rainer Ellinger is a microcomputer assembly language and hard-ware expert. One of his many skills is taking complex computing subjects and making them understandable to the average user.

ISBN 0-916439-44-5

A Data Becker book published by

You Can Count On
Abacus Software

P.O. Box 7219 Grand Rapids, MI 49510 - Telex 709-101 - Phone 616/241-5510