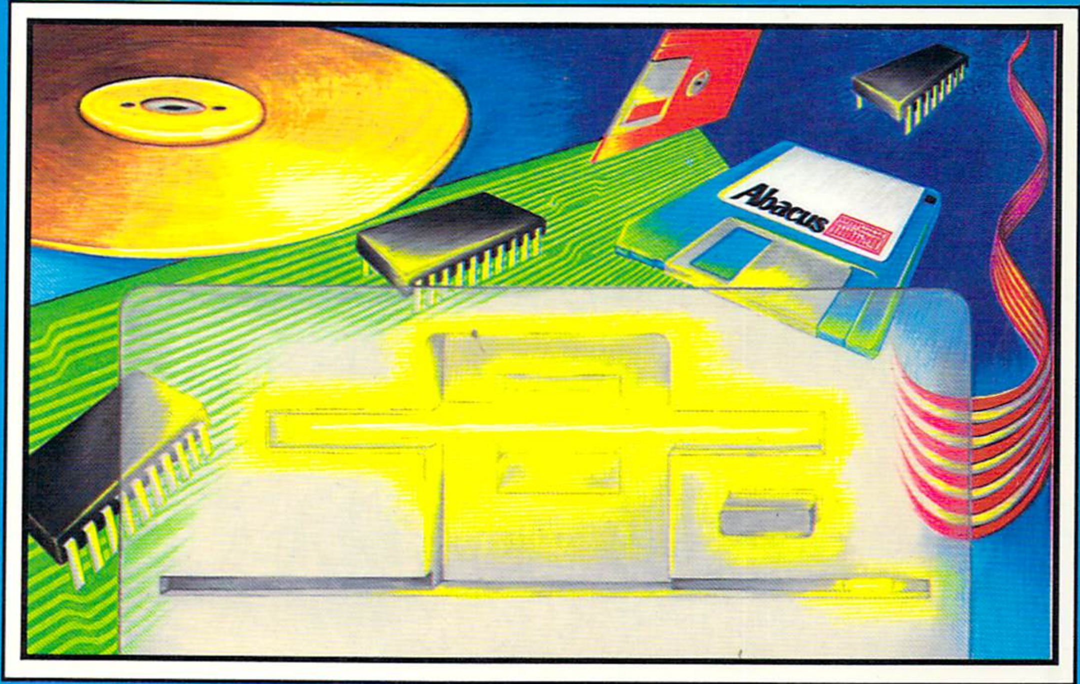
Amiga® Disk Drives Inside & Out

The most thorough coverage of Amiga Disk Drives ever.

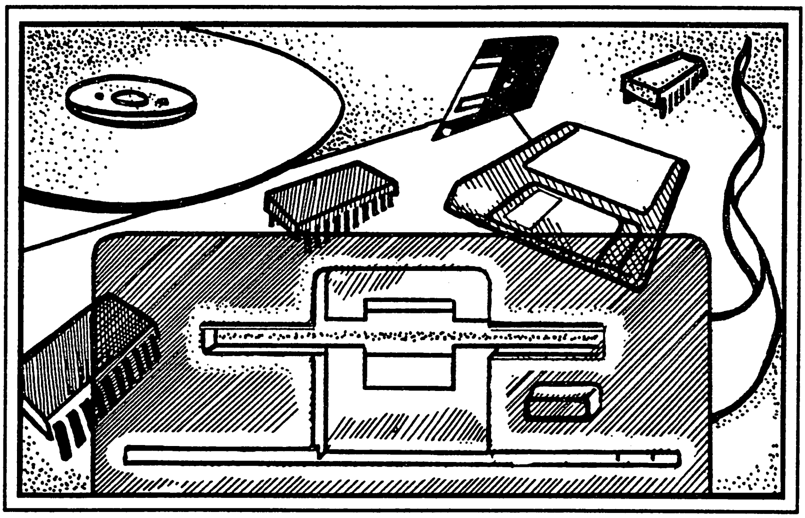


Abacus

A Data Becker Book

Amiga Disk Drives: Inside & Out

Grote Gelfland Abraham



Abacus

A Data Becker Book

First Printing, November 1988
Printed in U.S.A.

ISBN 1-55755-042-5

Table of Contents

Preface

The disk drive is an important part of the Amiga. Disk drives let you store data for later recall. Whether it's an address file, monthly expenses or a letter, you can save the data to a disk and load it back into memory later.

This book will help you understand what disk drives are and how they work. Whatever your level of knowledge as an Amiga user, ***Amiga Disk Drives Inside and Out*** gives you the information you need about Amiga disk drives.

This book contains descriptions of the disk drive operations used in AmigaBASIC, the Workbench and the `CLI` (Command Line Interface). In addition, you'll find information about direct access (programming the hardware and the operating system). Finally this book offers know-how about speeding up disk drive access, and a powerful disk copying program.

We feel that this book and the programs included will supply the reader with a good working knowledge of the Amiga disk drive.

The Authors

1.
Introduction

1. Introduction

Disk drives just didn't exist on Commodore's earlier computers (e.g., PET, VIC-20). Datasettes (cassette recorders) were the mass storage devices in these older machines; data was saved and loaded on standard audio cassettes. Disk drive interfaces became common in the later Commodore computers, but cassette interfaces still existed.

The old days are gone: Cassettes are too slow for data access on a 68000-based computer. The disk drive plays the central role in data storage on the Amiga. Every Amiga comes from the factory equipped with an internal disk drive. The disk drive is such a necessary piece of hardware that you can't get your Amiga started without inserting the Workbench disk.

We chose to keep this chapter as short as possible, so that you can get on to the main material of this book quickly. You should already be familiar with the basic concepts of using disks, even if that only consists of knowing how to insert the Workbench disk. See your Amiga manual or read Abacus' ***Amiga for Beginners*** for information about using your disk drive.

1.1 The disk

Disks are the storage medium used by disk drives. Without a disk, a disk drive is totally useless. The Amiga uses a 3.5 inch floppy disk. Disks of this size have been on the market for some time, but have only become popular in the last few years. IBM, Atari and Apple are just a few companies using 3.5 inch disks for data storage. The standard disk size for older home computers is the 5.25 inch floppy disk.

There is no one standard storage capacity for 3.5 inch disks. Each computer manufacturer uses his own method for storing information, so disk capacity varies from machine to machine.

The disk drives used in the Amiga have two read/write heads. Each head accesses a side of a disk. Commodore recommends that you purchase 2DD disks only. The 2 stands for "double-sided," the DD for "double-density." Double-sided double-density disks are sturdier than 1DD (single-sided double-density) disks.

If you use a 1DD disk, an error can occur on the second side of the disk during disk access or even during formatting. Manufacturers don't test the "B side" of 1DD disks for hardware errors. This is why 1DD disks are less expensive.

The solution: Spend the extra money for double-sided double-density disks. Don't use single-sided, double-density disks—you could lose data.

Formatting

Before an Amiga (or any computer) can use a disk for storing data, the disk must be *formatted*. This process prepares a disk for receiving information. Formatting converts the disk's magnetic media into the order specified by the computer's operating system.

The Amiga operating system (AmigaDOS) formats each side of a disk into 80 tracks of fixed width. These tracks appear in concentric circles around the center of the disk. Each track of each side in combination (e.g., track 1 top and track 1 bottom) is called a *cylinder*.

Each track can be divided into 11 *sectors*. Every sector can store 512 bytes (512 characters). If you multiply the values presented here, you'll find that an Amiga disk can hold about 880K:

```
2 Read/Write heads      multiplied by
   80 Tracks per head      multiplied by
      11 Sectors per track    multiplied by
         512 Bytes per sector
            = 901,120 bytes        (1 character = 1 byte)
```

Since 1K corresponds to 1,024 bytes, dividing 901,120 bytes by 1,024 gives us 880K. To this figure we must add another 28K for directories and the File Allocation Table (FAT). A 3.5 inch Amiga format disk can easily hold 180 typewritten pages, or more than 900,000 characters.

With a such a large storage capacity, data organization becomes very important. AmigaDOS lets you store data under specific names on the disk. To keep the data better organized, this data can be placed in different disk areas allocated for a set of files.

Files

A collection of data is called a *file*. The Amiga has a number of different file types. For example, a Tool or program file is an executable program. The Notepad on the Workbench generates a Project (text) file which requires a Tool for access (i.e., the Notepad). The term file is a generic term for data. It can be used interchangeably for referring to programs, text files, BASIC programs and more.

1.2 Suggestions about this book

We have a few suggestions that will help make your work in disk access easier and more enjoyable. We felt that we should mention these suggestions now, before you read any further.

Make a backup copy of any disks you plan to use during the course of this book. This includes the Workbench disk and Extras disk. Disk access commands in this book should only be performed on a backup disk, even if the command looks harmless. This is to avoid any accidents. One incorrectly typed program line can destroy a disk. You can make backup disks using the DiskCopy command in the `CLI` or the Duplicate item in the Workbench menu. When done making backups, store the original disks in a dust-free, non-magnetic place for safekeeping.

Make a backup copy of the optional disk for this book if you bought this disk. Copy this disk and store the original disk in a safe place. Files which belong together on the disk have been assigned to drawers. For example, look in the drawer named CH4 for any programs that are listed in Chapter 4. The programs listed in the appendices of this book are located in the drawer named Assembler, since they are assembly language programs.

2. Workbench disk drive functions

2. Workbench disk drive functions

The Amiga disk operating system (AmigaDOS) handles the Workbench's disk functions. An interface between the user and the machine is needed to convey disk commands to AmigaDOS.

When you first turn the Amiga on, it doesn't recognize mouse movement or keyboard access. The Workbench disk which comes equipped with every Amiga establishes that communication between the Amiga and user. After the Workbench loads you can perform disk operations using the mouse and pulldown menus. These menus and the mouse make computing very easy for the new user.

The Workbench has some disadvantages which should be mentioned. The worst is that you usually can't see every filename on the disk from the Workbench, because of unavailable graphic information. However, the Workbench is a reliable user interface, and many Amiga users don't use anything else (see Chapter 3 for an alternate interface).

2.1 Copying single files

Every Amiga user with only 512K of memory and a single drive knows how bothersome and time consuming it can be to copy a single file to another disk.

You drag the program icon of the file to be copied into the window or onto the icon of the destination disk. You'll have to change disks at least four times, depending on the size of the program. As everyone knows, frequent disk switching is hard on both the operator and the disk drive.

One alternative would be to buy an external disk drive. This is fine, but costs money. You could just put up with the disk switching, but it can get very inconvenient.

The RAM disk

There's a better way to copy files quickly and easily: Use the Amiga's RAM (Random Access Memory). You can set aside a section of memory to act as a fake disk drive. This imitation disk drive is called a RAM disk. A RAM disk works much faster than an internal or external mechanical drive. Of course even the RAM disk has a disadvantage: As soon as you turn the computer off, all data stored in the RAM disk at the time is erased forever. If you reset the Amiga by pressing <Ctrl><Commodore><Amiga> or <Ctrl> left<Amiga> right<Amiga> (depending on the Amiga model), this also destroys any data in the RAM disk. As long as the Amiga is under power and hasn't been reset, the RAM disk information is as safe as if it were on a disk. You should make a point of saving RAM disk data to a floppy or hard disk occasionally.

The RAM disk's capacity changes with the amount of data it contains. This is limited by the amount of memory available, unlike the floppy disk which can hold 880K. If you have memory expansion in your Amiga, the RAM disk can store more data. There's no absolute limit to the RAM disk's capacity. However, the Workbench always states that the RAM disk is full, even if you only store one byte in it.

If you already see a RAM disk icon on your Workbench screen, skip to the paragraph entitled Copying with the RAM disk on the next page.

The RAM disk isn't installed directly in some versions of the Workbench. We'll have to enter the `CLI` to create the RAM disk (more on the `CLI` in Chapter 3). Double click the `CLI` icon to open it (the program is normally in the System directory on the Workbench disk). Some versions of the Workbench let you hide the `CLI` using Preferences. If the `CLI` icon cannot be found, double click the Preferences icon. When the Preferences screen appears enable the `CLI`.

Click the Save gadget to save this data. Open the System drawer. The System window should display the CLI icon.

After starting the CLI, a window appears displaying the prompt (1>). Enter the following command sequence, pressing the <Return> key at the end of each line:

```
1> dir ram:<Return>
1> endcli<Return>
```

The dir ram: command calls the directory of the RAM disk. Since no RAM disk exists yet, AmigaDOS creates a RAM disk and displays a RAM disk icon on the Workbench screen. The Endcli command terminates the CLI, closes the window and returns to the Workbench screen. The RAM disk icon appears on the screen.

Copying with the RAM disk

To copy single files using the RAM disk, try this example. Look for the Clock icon (it may be in the Utilities drawer in some versions of the Workbench). Drag the Clock icon from the Workbench window to the RAM disk icon. The pointer changes to a wait pointer. The drive runs for a moment, then the wait pointer changes into the normal mouse pointer. Double click the RAM disk icon and you'll find the Clock icon inside. Now insert a disk in the internal drive and wait for its icon to appear. Drag the clock from the RAM disk window to the icon of the newly inserted disk. The Clock program moves to the disk in the internal drive.

To free up storage space in RAM select the Clock icon in the RAM disk and select the Discard item from the Workbench menu. A requester appears. Click on the OK to discard gadget to delete the clock from the RAM disk.

2.2 Deleting files

The Workbench has two ways of deleting files from the disk. One method is to click the selected file icon then select the Discard item from the Workbench menu. A requester appears to tell you "Warning: you cannot get back what you discard." If you click on the OK to discard gadget, the Workbench deletes the selected file from the disk.

Empty Trash

The second method consists of dragging the icon of the file to be deleted to the Trashcan icon. The file disappears from the screen. The program still exists; you've moved it to the Trashcan, which stores files like a drawer. Selecting the Empty Trash item from the Disk menu actually removes the files from the Trashcan. The advantage of this over the Discard item is that files which you accidentally moved to the Trashcan can be recovered from the Trashcan as long as you haven't selected Empty Trash. The Empty Trash item releases the memory area the files occupied on the disk.

2.2.1 Deletion protection for files

There are some features which prevent the deletion of data and programs. Select the icon of the Clock program with a single click. Now select the Info item from the Workbench menu. After the drive finishes running, an Info window appears on the screen. The Info window has a number of gadgets in it. The two string gadgets in the upper part of the window display the name and object type of the file. The Amiga currently has five object types:

Type	Sample Object
Disk	Workbench disk
Garbage	Trashcan
Drawer	System directory
Tool	Notepad
Project	Notepad

The object type determines whether additional indications appear in this string gadget. For example, Garbage or Drawer object types need no additional information. Disk object types require information such as total capacity, the number of blocks already occupied and the number of blocks still available, and the difference of the two numbers. Finally block size information appears: "Bytes per Block 512". Block capacity corresponds to a sector on disk.

Status

A Status field appears at the upper right of the Info window. The word Deleteable indicates that the file whose Info window currently displayed can be deleted from the Workbench. Place the mouse pointer on the gadget currently containing the word Deleteable and click once. The display changes to Not Deleteable. Now you can't delete this file from the disk using the methods described above. Click the Save gadget to make the clock undeleteable and exit the Info window.

Select the Clock icon and then select the Discard item from the Workbench menu. The requester appears; click on the OK to discard gadget. After the drive runs, the Workbench title bar displays the message "Error while removing clock:222." You can't delete the clock until you change the status back to Deleteable.

2.2.2 Autostarting a Project

Project (text or program code) icons can be copied back and forth on the Workbench screen. You can also execute a Project by double clicking its icon. The Project itself is not an executable program: It needs its corresponding Tool (main program) to execute. When you double click a Project, the Workbench looks for the Tool used to create the project. When the Workbench finds the Tool, it loads and executes the Tool then loads the Project into memory. If the Project is program code (e.g., AmigaBASIC) the Project executes.

How does the Workbench know which Tool created the Project? Create a short document (Project) using your Notepad (it's in the Utilities drawer of your Workbench disk). Save the Project and quit the Notepad. Close and reopen the Utilities drawer. Click on the Project's icon and select the Info item from the Workbench menu. The Info window appears.

Default Tool

The Default Tool string gadget contains the entry "sys:Utilities/Notepad". This means that the Notepad Tool was used to create this Project and can be found on the system disk (the disk with which you started your system) in the Utilities drawer. If you close the Info window and double click this Project, the Amiga checks to see if there is a Default Tool entry available. If this entry exists, the Amiga searches for the Default Tool, loads the Default Tool and loads the Project as well.

This autostarting capability can be useful with word processing programs or AmigaBASIC. Double clicking a Project is much easier than double clicking the Tool, waiting for the Tool to load, then double clicking or loading the Project.

There are two other ways of accessing a Project and its Tool. Try each example using the Project you created above using the Notepad:

1. Click once on the Project icon you created from the Notepad. Press and hold a <Shift> key and double-click on the Notepad icon.

2. Press and hold a <Shift> key. Click once on the Project icon you created from the Notepad. Click once on the Notepad icon. Release the <Shift> key; both objects should be highlighted. Select the Open item from the Workbench menu.

These last two options also work when no entry exists in the Default Tool string gadget. This lets you load files created by another word processing program directly into the Notepad, without changing the Default Tool string gadget

Tool Types

The Tool Types string gadget describes information which automatically passes to the indicated Tool. The Notepad Tool Types gadget lists the names of the font files loaded in while the Project loads. Unfortunately, you can't change Tool Types in non-Notepad Projects.

2.3 Tips & Tricks for the Workbench

The following are some tips and tricks which make working with the Workbench much easier on the user.

Selecting multiple files

Multi selection is of the best features on the Workbench. For example, if you want to delete a whole series of programs from a disk, press and hold a <Shift> key while clicking on every file icon you want deleted. Select the Discard item from the Workbench menu to delete all the highlighted file icons.

Multi selection can also be used to copy multiple files onto another disk.

Keyboard shortcuts

You actually don't use the mouse much when running a word processor or the `CLI`. However, if a requester appears, you must switch to the mouse to click on one of the gadgets.

You don't have to reach for the mouse. You can actually select requester gadgets from the keyboard. Pressing the key combination <Commodore><v> (or left <Amiga><v>) corresponds to the selection of the Retry gadget with the mouse. The key combination <Commodore><b> (or left <Amiga><b>) activates the Cancel gadget.

3.
The CLI

3. The CLI

The Command Line Interface (CLI) is another method of communicating with AmigaDOS. The CLI is much more powerful than the Workbench. It is also much more difficult to use than the Workbench. You must learn specific commands and command syntax. The CLI's access isn't as easy as selecting a menu item or double clicking an icon.

Maybe the CLI seems like a throwback to the early days of home computing. Not true: The CLI is part of the Amiga's true power, especially in multitasking.

The Workbench only allows very limited multitasking. For example, if you format a disk using the Initialize item from the Disk menu, the computer remains unavailable for any other tasks. However, using the CLI lets you format a disk while printing a document and writing a new document, all at the same time. The more you learn about the CLI, the less you'll probably use the Workbench.

3.1 The CLI's capabilities

To begin learning the CLI boot (start) the Amiga. Insert the Workbench disk. If the power's off, turn on the Amiga. If the power is already on, press <Ctrl> <Commodore> right <Amiga> (or <Ctrl> left <Amiga> right <Amiga>) to reset the computer. When the Workbench finishes loading, open the Workbench disk, then the System drawer. Double click on the CLI icon.

Only disk specific CLI commands appear in this chapter. Other commands mentioned earlier will not be repeated here (e.g., DiskCopy).

3.1.1 Starting disks with Install

During power-up, the Amiga automatically loads the Workbench when you insert the Workbench disk. If you insert a non-Workbench disk, the Amiga hand icon stays on the screen. The Amiga checks the *boot sector* of the inserted disk for information that indicates a boot disk. If this information doesn't exist, the Amiga displays the hand icon until you insert a Workbench or other bootable disk.

The Install command creates a bootable disk. It writes the information required by AmigaDOS into the boot sector, making the disk bootable. The Install syntax is as follows:

```
1> install DRIVE
```

"Drive" must be replaced with the drive specifier in which the disk is located. The following makes the disk in the internal drive bootable:

```
1> install df0:
```

You can use the Install command on a newly formatted disk which doesn't contain any files. However, if you reset the Amiga with this disk in the internal drive, the booting process is not completed properly. For example, only 60 characters fit into a display line. Also, the normal commands are available. This occurs because there are files missing from the disk. See Section 3.2 for the minimum files required for booting the Amiga.

3.1.2 Info

Don't worry if during a save process the requester appears with the "Disk Full" message, it is not a serious problem. You can insert a new disk and repeat the save process. Some programs, however, are very sensitive to a disk with too little storage space. You might get a "Guru Meditation" and lose all data in RAM. You should check disk space when using these types of programs. The `CLI` Info command displays a list similar to this:

```
Mounted disks:
Unit Size   Used    Free Full Errs    Status     Name
df0: 880K   1683    75   95%    0   Read/Write  Workbench

Volumes available:
Workbench [Mounted]
```

The "Mounted disks" heading contains information about the drives connected to the Amiga. The example above lists the contents of the internal drive (df0:) only. This information includes the disk size (880K); the number of blocks used (1683); the number of disk blocks unused (75). In addition you'll find the percentage of the disk full (95%); the number of defective blocks (Errs=0); the write protect status of the disk (Read/Write); and the disk's name (Workbench).

About Status

If the write protect on the disk is open, Info displays Read status (you can only read from the disk). If the write protect clip of the disk covers the opening, Info displays Read/Write status (you can read, write and delete disk files).

The "Volumes available" heading lists the names of the disks currently recognized by AmigaDOS. An additional [Mounted] indicates that this disk is currently available for access.

3.1.3 Protecting files

When you want a file protected against accidental deletion from the Workbench, you should select the Not Deleteable gadget from the Info window. The `CLI` performs the same function using the Protect command. The syntax of the command is:

```
1> protect [Filename] [Status]
```

Enter the name of the file to be protected for "Filename" and to which the information in "Status" applies. The best explanation is through another practical example using the Workbench clock:

```
1> protect clock rwed
```

Each of the four letters rwed stands for a special file attribute:

rwed

- *r* The file can be read by the program.
- *w* The file can be written to by the program.
- *e* The file can be started with execute.
- *d* The file can be deleted.

Unfortunately AmigaDOS in this case plays a little joke by ignoring the three flags besides the Delete flag (the d). This may be upsetting, but true. An example:

```
1> protect clock
```

This input protects the Workbench clock from accidental deletion since the Delete flag was not set. The remaining three flags are of no concern since they are not considered by DOS. The command:

```
1> protect clock rwe-
```

delivers the same result. Of course the status set in the `CLI` can be read. The List command displays the file's status. Here's the default status of the clock as displayed by the List command:

```
1> list df0:
...
clock                18000 rwed 07-Oct-86 15:27:27
...
```

Note that the information displayed includes the filename, the flags, the creation date and time. This example shows that all four flags are set. You can delete the Clock program in this state. The following input disables the delete status:

```
1> protect clock rwe-
```

Entering List again displays the following line for the clock:

```
clock       18000 rwe- 07-Oct-86 15:27:27
```

The file cannot be erased on the disk from either the Workbench or `CLI`.

The date and time items will only be correct if you set the current date and time in Preferences before saving or copying, or if the Amiga is equipped with a battery backed realtime clock.

3.1.4 The DiskDoctor

The 3.5 inch disks protect themselves quite well against the outside world (dust, fingerprints, etc.). However, the Amiga can't decode disk data sometimes. In that case, the Amiga displays a requester indicating the error on the disk (e.g., disk structure corrupt). Another requester suggests that you use the DiskDoctor to fix the disk structure.

The DiskDoctor

The DiskDoctor is a small program which can only be called from the CLI. This program doesn't perform miracles. It can provide a valid structure for a disk. It may also be able to rescue most of your corrupted disk data. This is possible thanks to the high *redundancy* (repetition) rate of information on one disk. For example, if damage occurs on track 18 of a Commodore 1541 format disk, both the directory and the file pointers are lost. AmigaDOS spreads directory information over the entire disk so that an extreme loss of data cannot occur quickly. The result is slow directory output since the read/write heads must travel back and forth to collect the desired information. The syntax of the DiskDoctor command is:

```
1> diskdoctor [drive]
```

The drive number for any connected 3.5 inch drive can be used (e.g., df0:).

At the prompt, insert the disk into the drive and press the <Return> key. The Doctor goes to work. First the program reads the file information sequentially from each cylinder. During this time DOS determines the locations of *hard errors* and tells the user the track number and side number:

```
Hard error Track 29 Surface 0
```

Hard error

A hard error can occur from mechanical defects (scratches, dirt, etc.) or from errors in the track formatting. A simple formatting procedure may be all it takes to make the disk useful again. Allow the DiskDoctor to finish its examination.

During the second run, the DiskDoctor displays the names of all files and directories readable from AmigaDOS. The screen displays what the DiskDoctor is doing with the file (Replacing/Inserting file/dir xxx). Badly damaged files are removed automatically, as seen in the line below:

```
Attention: Some file in directory xxx is unreadable and
has been deleted
```

If a file contains data which are only partially unreadable, the file remains on the disk but a warning appears:

```
Warning: File xxx contains unreadable Data
```

Finally the DiskDoctor asks if you want the defective files deleted from the directories:

```
Delete corrupt files in dir xxx ?
```

If you have worked with a disk monitor, some items can be rescued. Otherwise it may be best to remove the defective files.

The last message of the DiskDoctor asks the user to copy all saved files to a new disk and format the disk which was processed by the DiskDoctor. The formatting routine determines whether the hard errors that occurred can be removed.

3.1.5 AddBuffers

The Amiga allocates a 25K memory buffer for each connected disk drive. This area acts as interim storage between the disk drive and the Amiga. All data sent from the disk first goes to the buffer before being processed in memory. Sometimes when you enter the Info command for the first time, the Amiga loads information from the disk. If you enter Info again, disk access may not take place, since the data required is still in the buffer.

The size of this buffer can be increased using the AddBuffers command. Since system memory decreases when you increase buffer size, you should be careful in your buffer allocation. You have a choice: Big buffer or high speed.

Here's a sample call:

```
1> AddBuffers df0: 20
```

The above call assigns drive df0: 20 blocks (=10K) more of buffer memory. Individual programs which are called sequentially (e.g., CLI commands) and whose sizes don't exceed 10K can be retained in memory without disk access. This remains in effect until another disk access overwrites the commands in the buffer.

3.1.6 Using the RAM disk with the CLI

CLI users who have only one drive may find it a nuisance that the CLI can only read its commands from the Workbench disk. If you want to examine the directory of any other disk with the dir command, the Amiga requests the Workbench disk. The reason for this phenomenon is simple: All CLI commands are nothing more than short programs normally stored on the Workbench disk. If you insert another disk in the internal drive, the Amiga cannot execute the CLI command dir df0:. It demands the Workbench disk. If you answer the request, the CLI displays the Workbench disk's directory instead of the desired directory. Some tricks can help here. If you only want to read the directory of the foreign disk once, enter the following command while the Workbench disk is in the drive:

```
dir ?
```

The Dir command loads without executing. Instead a line appears on the display which contains information about the correct syntax of the Dir command. This line of syntax is called the ***argument template***:

```
DIR,OPT/K:
```

The cursor waits for input. Remove the Workbench disk and insert the disk you want to view. Enter df0: and press <Return>. The CLI displays this directory. This method of using the question mark can be applied to all CLI commands.

There are other ways of applying CLI commands to other disks. The RAM disk can help. Reset the Amiga, open the CLI and enter the following lines:

```
1> cd df0:
1> makedir ram:c
1> copy c: to ram:c all
1> assign c: ram:c
```

This set of commands copies all the CLI commands to the RAM disk and makes the RAM disk the drive to search for these commands.

Problem: If you use the above four commands on an Amiga with 512K RAM, you won't have much RAM left. In fact, you won't be able to load any large programs. The solution: Copy only those commands you think you'll need to the RAM disk. Reset the Amiga, open the CLI and enter the following lines:

```
1> makedir ram:c
1> copy c/dir to ram:c
1> copy c/list to ram:c
1> copy c/cd to ram:c
1> copy c/copy to ram:c
1> copy c/delete to ram:c
1> copy c/type to ram:c
1> .
   .
   .
1> assign c: ram:c
```

Replace the above periods with other commands you want copied into the RAM disk. The commands above show the most often used CLI commands. Commands used less frequently only occupy a little memory—these can be loaded from the Workbench disk into the RAM disk at any time. Insert the Workbench disk in the internal disk drive and enter the following command sequence:

```
1> copy df0:c/[CLI-command-desired] to ram:c
```

Typing these commands every time you start your Amiga takes time and can get to be boring. Instead you can put the commands into a *script file* and use the script file to create the CLI-based RAM disk.

Script files

A script file is a normal text file which can be created by any word processing program. It contains a series of CLI commands which are executed in sequence. You start a script file by entering the Execute command, followed by the script file's name. This saves typing the input for often used CLI command sequences such as RAM disk installation. The startup sequence which executes immediately after booting the Amiga is really a script file.

You can create a script file with any word processing program that can save data as an ASCII file (e.g., BeckerText). The editor ED contained in the C directory of the Workbench disk is all you need to write a script file. Call ED by entering:

```
1> ed [Filename]
```

Let's create the RAM disk routine as a script file. Enter:

```
1> ed ram-disk
```

Enter the CLI commands listed above for creating a RAM disk. Press <Esc><x> to save the script file and exit ED. Reset the Amiga, open the CLI and enter the following to run the script file:

```
1> execute ram-disk
```

Modifying the startup sequence gives you the CLI-based RAM disk available when you reset. Change directories and invoke ED as shown below:

```
1> cd df0:
1> ed s/startup-sequence
```

The desired script file appears on the screen. Insert the lines below before the Loadwb command:

```
dir ram:
execute ram-disk
```

Press <Esc><x> to save the change and exit ED. After every restart the Amiga installs a new RAM disk automatically.

3.1.7 Path

When you type a command into the `CLI`, AmigaDOS searches on the disk for a program of the same name in a specific order. The search begins in the current directory. If DOS doesn't find the command there, the C directory is searched next. If DOS still can't find the command it displays an "Unknown command" message.

The Path command prevents this error. This command tells DOS to automatically search additional directories for the command which was entered.

An example is the startup sequence of the Workbench disk. This sequence looks like this in some versions of the Workbench:

```
echo "Workbench disk (Version 1.2/33.43)"
echo " "
echo "(Date and Time can be set with 'Preferences')"
if EXISTS sys:system
    path sys:system add
endif
BindDrivers
setmap d
LoadWb
endcli > nil:
```

The "if EXISTS" command searches the boot disk for the System directory. If this directory exists, the Path command includes this directory in the list of directories to be searched. This is why the DiskCopy command can be called without having to indicate that it is in the System directory: AmigaDOS automatically checks this directory when looking for a command.

If you enter Path without arguments (parameters), it displays the directory names in the current path. AmigaDOS searches in the order indicated. The Path Reset command deletes all pathnames previously entered. Only the current directory and the directory assigned to device C: are searched (see Section 3.1.9 for a description of device C:).

3.1.8 DiskChange

AmigaDOS immediately recognizes every disk change in the internal and external drives. Since the 3.5 inch drives report disk changes automatically to the operating system, it's impossible to write data to the wrong disk. However, the larger 5.25 inch drives act differently.

Most 5.25 inch disk drives don't tell the Amiga that a disk change has occurred. This could cause damage to the disk currently in the drive due to the Amiga adding data to what it thinks was the disk previously in the drive.

The DiskChange command should be entered after every 5.25 inch disk change. The following command tells the operating system that a different disk has been inserted in drive df1:

```
1> diskchange df1:
```

3.1.9 Assign

The operating system of the Amiga uses a colon to differentiate between device names and filenames. (e.g., df0:, df1: and prt: are device names). The Assign command permits device name assignment to directories or filenames. Assign entered without any parameters displays the current assignments. Here is an example, your display may differ:

```
1>ASSIGN
Voulmes:
RAM DISK [Mounted]
Workbenchg 1.3 [Mounted]

Directories:
CLIPS        RAM DISK:clipboards
ENV          RAM DISK:env
T            RAM DISK:t
S            Workbench 1.3:s
L            Workbench 1.3:l
C            Workbench 1.3:c
FONTS        Workbench 1.3:fonts
DEVS         Workbench 1.3:devs
LINS         Workbench 1.3:libs
SYS          Workbench 1.3:

Devices:
PIPE AUX SPEAK NEWCON DF2
CON RAM
1>
```

Assume that you have a text file named Peter inside the Letters directory. This directory is inside the Text directory in drive df0:

```
df0:Text/Letters/Peter
```

You want to copy this letter to several disks. It would be very time consuming to input the following line for every Copy command:

```
copy df0:Text/Letters/Peter to Copy1
```

The Assign command assigns this directory structure a shorter name:

```
assign Orig: df0:Text/Letters/Peter
```

After entering this command the Copy command can be abbreviated as follows:

```
copy Orig: to Copy1
```

The "Orig:" is now shorthand for "df0:Text/Letters/Peter". This method of assigning device names to paths can save you a great deal of typing.

If you enter the Assign command without arguments (parameters) it returns a list of assignments currently recognized by AmigaDOS. This is handy if you can't remember which device names are available.

The pseudo device C: is also an assigned device name. The Path command used previously showed C: as the last directory entry to be searched for `CLI` commands. The C directory on the Workbench disk is normally searched automatically. That's why AmigaDOS demands this disk when you enter a `CLI` command. If you copied the `CLI` commands into the RAM disk as discussed previously, "assign C: RAM:c" tells DOS to search the C directory of the RAM disk for `CLI` commands.

To disable an assignment, enter Assign and the name alone:

```
assign Orig:
```

The command above disables the "Orig:" device.

3.2 Interactive directory

Even though Amiga disks have 880K of memory, they get full eventually. Disks capable of booting (like the Workbench) have less storage capacity since they already contain many drawers. Many programs in these directories are seldom used and take up valuable space.

The Dir command lists the contents of the Workbench disk. Entering Dir without arguments displays all directories on the disk at once. The sequence below lists all directories interactively:

```
1> dir opt ai
```

This option lists all directories, including their individual files, and displays a question mark prompt after each entry. The directory wants you to respond (interact). You have three options in interactive mode when displaying all of the files:

<del><Return>	Type the word "del" to delete the file.
<t><Return>	Display the contents of a text file.
<Return>	Display next file/directory.

This command helps you find unnecessary files on the Workbench. Start with the printer drivers. Only the drivers which fit the attached printer are needed. All other drivers can be erased (enter del and press <Return>). Most users would not want to select a foreign keyboard layout on the Amiga. Erasing all the keyboard drivers from the Devs/Keymaps directory except "usa0" and "usa1."

As soon as you buy a decent word processor, delete the Notepad from the Workbench disk. Delete the Expansion drawer if you don't own a hard disk drive or other exotic peripherals.

The following files can also be removed if you don't need them:

Path	Condition for deletion
devs/serial.device	Printer not on serial port
devs/parallel.device	Printer not on parallel port
devs/Mountlist	No special peripherals attached (e.g., 5.25 inch drive)
fonts/...	(depends on applications)
demos/...	(usually can be removed)
Clock	(depends on applications)
Calculator	(depends on applications)

Let's use the Utilities drawer as an example of how you can operate the Dir command interactively. Enter the following:

```
1> dir opt i
```

Press the <Return> key until the Utilities drawer (directory) is displayed on the screen. Enter <e> to enter this directory. To delete the Calculator, enter the word <del><Return> when the words calculator and calculator.info are displayed. The entire disk can be searched in this way for unnecessary files.

Here is an example of the display, your display may differ slightly:

```
1>dir df0: opt i
     Trashcan (dir) ?
     c(dir) ?
     Prefs (dir) ?
     System (dir) ?
     l (dir) ?
     devs (dir) ?
     s (dir) ?
     t (dir) ?
     fonts (dir) ?
     libs (dir) ?
     Empty (dir) ?
     Utilities (dir) ?e
       .info ?
       Calculator ? del
       Calculator.info ?del
...
...
..
1>
```

3.3 Tips & Tricks for the CLI

There are a lot of tips and tricks to make your CLI sessions more efficient. The following section presents some of these.

Keyboard shortcuts in the CLI

The table below shows some shortcuts available in the CLI:

Keys	Function
<Ctrl><D>	Interrupts script file.
<Ctrl><C>	Terminates a command which is executing.
<Ctrl><L>	Clears the screen.
Any key	Stops text output.
Backspace	Continue output.
<Commodore><v>	Click on Retry gadget.
<Commodore><b>	Click on Cancel gadget.

Depending on the Amiga model, you may have either a <Commodore> key or a left <Amiga> key.

Abbreviating CLI commands

If the CLI commands become too long (e.g., Execute, Addbuffers, BindDrivers, etc.) the CLI commands can be renamed with the Rename command. If you use script files frequently, the Execute command can be renamed to "ex" which saves some typing. The following example illustrates the renaming procedure:

```
1> rename c/execute to c/ex
```

You can now type "ex startup-sequence" instead of "execute startup-sequence". Try the abbreviation "ren" for the Rename function. The command for this would be as follows:

```
1> rename c/rename to c/ren
```

Other abbreviations are easily implemented:

Command	Abbreviation
dir	d
copy	c
delete	del
type	t
run	r
etc.	

This list can be expanded according to the user's needs. Avoid abbreviations that don't even remotely resemble the command word; the new word will be harder to remember.

4. Programming in BASIC

4. Programming in BASIC

Many computer novices were introduced to the world of computer languages through BASIC. Some old hands make fun of the language because it is too slow and convoluted. Times have changed. From the orginal spaghetti-code BASIC faster and more productive BASIC dialects have been developed. These new versions can be used in the professional development of programs. ABasic, which was delivered with the first Amigas, didn't offer half the capabilities of structured and module oriented programming of the modern AmigaBASIC. The Amiga, with its fast 68000 processor, gives the old C64 experts a new trust in BASIC programming. Where a "FOR i=1 TO 10000" required at least ten seconds on the old C64, the Amiga can count this down in a mere two seconds. This is not the only proof of speed which can be achieved on the Amiga.

4.1 LOAD, SAVE and Co.

The user who has loaded or stored a BASIC program is familiar with the two commands LOAD (load program) and SAVE (store program). AmigaBASIC also offers some other capabilities:

To have a "lightning" start without having to load the program and then RUN it, the following variation is used:

```
RUN "Program"
```

The program is loaded from disk and started immediately. If the user knows before starting AmigaBASIC which program will be used, the icon of the desired program can be double clicked. This starts Amiga-BASIC first and then automatically loads and executes the program which was clicked. This is also possible from the `CLI`:

```
1> RUN amigabasic "Program"
```

This input does the same as a direct start from the Workbench.

Several alternatives also exist during saving. If the normal SAVE command is followed by a comma, the command can be enhanced with three different options.

```
SAVE "Program",a
```

a.) Option "a" stores the program in ASCII format. In this format it can be loaded by any word processing program and edited further.

```
SAVE "Program",b
```

b.) If option "b" is used, the program is stored in binary. A program stored under this option requires less storage space on the disk than a program stored under the "a" option. It can also be loaded much faster. An attempt to load such a file into a word processing program fails since the BASIC commands are not stored as ASCII text, but in an abbreviated format.

```
SAVE "Program",p
```

p.) The "p" option is the last and surely the most interesting possibility of storing a program on disk The program is encoded (protected) by this option and stored on disk. It can no longer be edited, only started. Any attempt to list the program results in an error message. This option is of interest to those users who want to prevent a listing of a program.

Since a program stored with "p" cannot be modified a backup copy should be made of the listing before using the "p" option for saving. Loading and attempting to save the program again using the "a" option results in an error message. For this reason extreme caution should be used.

Another interesting capability offered by AmigaBASIC is the consecutive loading of several programs into the BASIC working memory. The MERGE command is used to do this. It can be used in direct mode and also in the program itself. Several routines (for example an Editor) can be stored as programs and when needed merged into other programs. Such a program is always added at the end of the programs in memory. It cannot be merged at any desired location.

4.2 Files in AmigaBASIC

In the preceding chapters the concept of a file was discussed. For the following sections two additional concepts must be explained:

The data record

If a computer file is compared with a card file, the individual cards in the card file correspond to the data records in the computer file. The data records of an address file, for example, contain the complete address of a certain person.

The data field

Every card in a card file is normally subdivided into several fields. For the address file box, the fields can be designated as follows: First Name, Last Name, Street, City, Phone. Every card in this file has five fields. In a computer file such subdivisions of a data record are called data fields.

These concepts should be thoroughly understood since they are frequently used in connection with data processing. If you run into trouble, remember the card file:

```
file                            <->   card file
contains Data records           <->   file cards
subdivided into Data Fields     <->   fields
```

The important role of files in AmigaBASIC is discussed in the next sections.

4.2.1 File types in AmigaBASIC

How would an address file be constructed? The Amiga has two different file types. First, sequential files which are characterized by simple handling. Second, the random access files which require more work, use more space, but are far superior to sequential files in their capabilities. Each of the two file types are useful for different applications which are discussed later.

4.2.2 A sequential file is created

First start AmigaBASIC. A sequential file can be "opened" with an almost limitless choice of names. The limitations are that some characters cannot be used when naming the file. These characters have special significance in AmigaDOS. These are:

: Determines drive (for example df0:, ram:).
/ Determines directory levels (drawers).
Used for search criteria in the `CLI` and some programs.
? Used for search criteria in the `CLI` and some programs.

It's possible to create a file named Example/file, but AmigaDOS interprets it to mean that in the Example drawer the file "file" should be addressed. A file with the name example-file causes no problems. If a colon occurs in the filename, AmigaDOS interprets the character string before the colon as a disk or device name.

To avoid misunderstandings a single filename is used in the following examples. We will create an address file named example-file. Input the following text into the LIST window of AmigaBASIC:

```
OPEN "example-file" FOR OUTPUT AS #1
```

Let's examine the syntax of the command sequence for opening a sequential file:

"OPEN" means what it says and is the key word in this command sequence. Then the file is named, in this case example-file. The OPEN command continues with direction of the information flow. This is an indication if the file should be written (FOR OUTPUT) or read only (FOR INPUT). Since data is first written into the file, it is opened for writing (FOR OUTPUT). Finally the command is given a file channel number which can have a value between 1 and 255. The file channel is important since all following commands for writing and reading refer to the file channel.

Continue adding to the program with the following lines:

```
DataEntry:
INPUT Person$
INPUT Street$
INPUT City$
```

Each one of these input commands accepts a data field from the keyboard. All three data fields together produce a data record.

```
PRINT #1,Person$
PRINT #1,Street$
PRINT #1,City$
```

The PRINT command

An interesting point has been reached in the development of the sequential file. The PRINT command is familiar from normal text output on the screen. Since file channel #1 was opened, the PRINT #1,... command does not output the following data on the screen, but sends them directly to the disk to the name of the file (example-file) opened on channel #1. Each of the three PRINT commands writes a data field to the disk.

Add the following:

```
INPUT "more";more$
IF more$="Y" THEN DataEntry
CLOSE #1
END
```

The INPUT command

The INPUT command asks if any additional data is input and stored, or if the file should be closed with CLOSE #1 and the program terminated. If another input is desired (y) the program branches again to the data input area (Label: DataEntry), and the entire procedure is repeated. Newly input data is appended to the previously input data. For this reason it is called a sequential file.

The CLOSE command

It is important to place the CLOSE command at the end of the program to close the file, or the integrity of the data in the file cannot be assured. The disk drive won't run after the completion of every data record. The data is first placed in a buffer until an access to the drive is justified. At the end of the program there may still be data in the buffer which has not yet been written to disk. The CLOSE command causes the data remaining in the buffer to be written to the file and the channel closed.

This completes half of the address file. Now you'll want to read the stored data and examine it. The following program section can be used:

```
OPEN "example-file" FOR INPUT AS #1
DataReader:
INPUT #1,Person$
INPUT #1,Street$
INPUT #1,City$
PRINT Person$
PRINT Street$
PRINT City$
INPUT "more";more$
IF more$="Y" THEN DataReader
CLOSE #1
END
```

Since this time data is only read from the disk, the file is opened with "FOR INPUT". The previous PRINT command redirected the output of the data to the disk and the INPUT #1,... command gets the data from the disk, not from the keyboard.

The program lacks one more item. As soon as an attempt is made to read additional data, the Amiga reports a "Read Past End" error. To avoid this error, a command sequence is added to check for the end of the file. The modified program appears as follows:

```
OPEN "example-file" FOR OUTPUT AS #1
DataEntry:
INPUT Person$
INPUT Street$
INPUT City$
PRINT #1,Person$
PRINT #1,Street$
PRINT #1,City$
INPUT "more";more$
IF more$="y" THEN DataEntry
CLOSE #1
OPEN "example-file" FOR INPUT AS #1
DataReader:
IF EOF(1)<>0 THEN End
INPUT #1,Person$
INPUT #1,Street$
INPUT #1,City$
PRINT Person$
PRINT Street$
PRINT City$
INPUT "more";more$
IF more$="Y" THEN DataReader
End:
CLOSE #1
END
```

The new function EOF() checks to see if there is another data record after the one just read. If there is not, a value other than zero is returned. The 1 in EOF(1) refers to the file channel number. EOF stands for End Of File.

4.2.3 Enlarging a sequential file

This should make the small address file complete. What happens if additional data is added? Since a sequential file always adds data directly to the beginning of the file, the stored data is overwritten. To prevent this, the APPEND mode (designation: "A") must be used. This appears as follows:

```
Enlarging:
OPEN "example-file" FOR APPEND AS #1
INPUT Person$
INPUT Street$
INPUT City$
PRINT #1,Person$
```

```
PRINT #1,Street$
PRINT #1,City$
INPUT "Append more";more$
IF more$="y" THEN Enlarging
CLOSE #1
END
```

This solves the problem. Some functions such as sorting and searching our address database are still missing, but for purposes of training this program is adequate. Let's examine a complete listing of a small address file management program.

Program 1:

```
'Program 1:

more:
INPUT "(R)ead next, (I)nput, (E)nd"; action$
IF action$="r" THEN DataReader
IF action$="i" THEN DataEntry
IF action$="e" THEN END
GOTO more

DataEntry:
CLOSE #1:
count=0
OPEN "example-file" FOR APPEND AS #1
INPUT "Name";Person$
INPUT "Street";Street$
INPUT "City";City$
PRINT #1,Person$
PRINT #1,Street$
PRINT #1,City$
CLOSE #1
GOTO more

DataReader:
IF cnt=0 THEN OPEN "example-file" FOR INPUT AS #1:cnt =1

IF EOF(1)<>0 THEN PRINT "No more records!": GOTO more

INPUT #1,Person$
INPUT #1,Street$
INPUT #1,City$
PRINT "Name",Person$
PRINT "Street",Street$
PRINT "City",City$
GOTO more
```

4.2.4 A Random access file

The simple handling of the sequential file has some disadvantages. Every addition of a data set increases the size of the file. If a certain data set is accessed, the entire file must be searched from the beginning for this entry. Even though AmigaBASIC is fast, a search of this nature through large files can take several minutes. A sequential file is therefore not suitable for a large address file. It is only useful if all the data stored is accessed at the same time, such as in program files where all the data is loaded sequentially into the working memory of the computer.

AmigaBASIC has a more suitable file type for the address file: the random access file, also called random file.

This file type permits random access to certain data records and their fields without searching a long sequential list. AmigaBASIC provides very powerful commands to access these types of files. The syntax for opening a file of this type is as follows:

```
OPEN "R",#1,"example2",40
FIELD #1,10 AS Number$,20 AS Description$,10 AS Price$
```

The OPEN command is the same one used in sequential files. The "R" designates the mode for both reading and writing random files (R stands for random access). The filename shouldn't present a problem. The only new item is the designation of a fixed data record length. Once it is set, it cannot be changed later. Therefore the number, in this example 40, should be considered carefully. You must designate a length that allows the input of the longest piece of information you'll want to use.

At first glance the second new command sequence, which starts with the FIELD command, appears somewhat difficult, but it's easily understood.

Following the FIELD command is the file channel number (#1). The variables, which contain the information to be stored on the disk are assigned a maximum length. The total of these is the exact length of the data record. It is best to consider the whole matter on the basis of an example. This time an inventory control file is created to illustrate the advantages of a direct access file:

```
FileName$="example2"
OPEN "R",#1,FileName$,40
FIELD #1,10 AS Number$,20 AS Description$,10 AS Price$
DataEntry:
INPUT "Stock-Number";nr
INPUT "Name";na$
INPUT "Price ($)";dollars
```

```
LSET Number$=MKS$(nr)
LSET Description$=na$
LSET Price$=MKS$(dollars)
PUT #1,nr
INPUT "more";more$
IF more$="y" THEN DataEntry

DataReader:
INPUT "Stock-Number";nr
GET #1,nr
PRINT "Name",Description$
PRINT "Price ($)",CVS(Price$)
INPUT "more";more$
IF more$="y" THEN DataReader
CLOSE #1
END
```

Let's take a moment to become familiar with the new commands. First, the LSET command. The FIELD command determines the maximum number of characters allowed in each field and a name is assigned to every data field. The INPUT command does not limit the length during input of data, so the input is first put into auxiliary variables "nr", "na$" or "dm." The assignment to field variables occurs through the LSET command. If the input is smaller than the maximum size allowed, the variable is padded on the right with blanks. All entries are left justified (LeftSET). If the input is longer than allowed, it's cut off on the right. Therefore every field variable ends up being the prescribed length.

The MKS$ function converts a number into an equivalent representation of text. The LSET command always expects a text variable so a number must first be converted before it can be stored in the proper field variable.

The PUT command finally stores the new data record on the disk. The data record number and channel number are both required. This is a numerical expression through which the data records are differentiated from each other. A data record number can only be assigned once. In this demonstration program the stock number serves this function.

The reading process now starts in reverse order. The command which reads the selected data set into the field variable is GET. The CVS command returns the "Price$" variable from text back into numbers.

Unlike the sequential file it is not a problem to add additional data records to an existing file. This is the advantage of the random access file because every data record has its own identity number under which it was stored and under which it can be read again. No specific sequence of data record numbers is required. Any desired number can be read.

The random access file is almost complete now. However, there is one problem to solve. The Amiga is not happy trying to read non-existent data from the disk. If data that doesn't exist is called for, it simply reads an arbitrary data record. The solution to this problem is discussed in Section 4.3.

Admittedly this concept is more difficult to understand than sequential files, but more can be accomplished with random access files. Besides the inventory control example with its stock numbers, prices and the other items, there are thousands of applications for this file type. Now we'll go on to the next section where some fun awaits.

4.2.5 Mini Data–The complete project

```
'Mini Data V1.0 © 1987 by GroSoft¶
¶
CLEAR,35000&¶
¶
Arrays:¶
DIM Enter$(15),Maske$(15),Search$(15)¶
¶
SCREEN 1,640,200,2,2¶
WINDOW 1,"Mini Data V1.0",,21,1         ¶
¶
FOR i=1 TO 10¶
MENU i,0,1,""¶
NEXT i¶
¶
MENU 1,0,1,"Mini Data"¶
MENU 1,1,1,"Open file        F1"¶
MENU 1,2,1,"New File         F2"¶
MENU 1,3,1,"Quit Mini Data   F3"¶
¶
MENU 2,0,1,"Search"¶
MENU 2,1,1,"Select  F4"¶
¶
MENU 3,0,1,"Mask"¶
MENU 3,1,1,"Mask change F5"¶
¶
MENU 4,0,1,"Printer"¶
MENU 4,1,1,"Print record   F6"¶
MENU 4,2,1,"Print file     F7"¶
¶
MENU 5,0,1,"Sort"¶
MENU 5,1,1,"Criterium  F8"¶
¶
COLOR 2,0¶
LOCATE 19,5:PRINT "File :"¶
LOCATE 20,5:PRINT "Record :"¶
¶
Buffer¶
COLOR 1,0¶
¶
MainLoop:¶
```

```
MENU ON¶
ON MENU GOSUB MenuBar¶
BREAK ON¶
ON BREAK GOSUB Interruption¶
ON ERROR GOTO Problem¶
ac$=""¶
ac$=INKEY$¶
IF ac$ ="" THEN MainLoop¶
IF ac$=CHR$(129) THEN FileOpen¶
IF ac$=CHR$(130) THEN NewFile¶
IF ac$=CHR$(131) THEN MiniDataQuit¶
IF MiniFile=1 THEN ¶
  IF ac$=CHR$(132) THEN Searcher¶
  IF ac$=CHR$(133) THEN MaskChange¶
  IF ac$=CHR$(134) THEN PrintRecord¶
  IF ac$=CHR$(135) THEN MiniFilePrint¶
  IF ac$=CHR$(136) THEN SortRoutine¶
  IF ac$=CHR$(31) THEN PrevRecord¶
  IF ac$=CHR$(30) THEN NextRecord¶
  IF ac$=CHR$(13) THEN DataEntry¶
  IF ac$=CHR$(28) THEN FirstRecord¶
END IF¶
GOTO MainLoop¶
¶
MenuBar:¶
Menue=MENU(0)¶
MenuPoint=MENU(1)¶
IF Menue=1 THEN ON MenuPoint GOTO
FileOpen,NewFile,MiniDataQuit¶
IF MiniFile=1 THEN¶
  IF Menue=2 THEN ON MenuPoint GOTO Searcher¶
  IF Menue=3 THEN MaskChange¶
  IF Menue=4 THEN ON MenuPoint GOTO
PrintRecord,MiniFilePrint¶
  IF Menue=5 THEN SortRoutine¶
END IF¶
RETURN¶
¶
FileOpen:¶
text$=""¶
LOCATE 22,5:PRINT "Please enter filename: "¶
Buffer¶
TextDataEntry 30,22,23,24,text$¶
LOCATE 22,5:PRINT SPACE$(70)¶
IF text$="" THEN MainLoop¶
ActualMiniFile$=text$+".MiniFile"¶
MiniFile=1:GOSUB MenuOn¶
CLOSE #1¶
quantity=0:nr=1:text$=""¶
OPEN "R",#1,ActualMiniFile$,730                 ¶
FIELD #1,10 AS a$,720 AS b$¶
LOCATE 19,12:PRINT SPACE$(70)¶
LOCATE 19,12:PRINT ActualMiniFile$¶
GOSUB Separate¶
GOSUB MaskLoad¶
GOTO FirstRecord¶
```

```
¶
NewFile:¶
text$=""¶
LOCATE 22,5:PRINT "Name of the file :"¶
TextDataEntry 22,22,23,24,text$¶
LOCATE 22,5: PRINT SPACE$(70)¶
IF text$="" THEN¶
  LOCATE 22,28:PRINT "Procedure terminated!"¶
  GOSUB Pause¶
  GOTO MainLoop¶
END IF¶
IF INSTR(text$,":")<>0 THEN¶
  LOCATE 22,19:PRINT "Please use the internal drive
only."¶
  GOSUB Pause¶
  GOTO NewFile¶
END IF¶
CLOSE #1:ActualMiniFile$=""¶
quantity=0:nr=1¶
text$=text$+".MiniFile"¶
ActualMiniFile$=text$¶
GOSUB MenuOn¶
OPEN "R",#1,ActualMiniFile$,730¶
FIELD #1,10 AS a$,720 AS b$              ¶
LSET a$=CHR$(1)¶
LSET b$=CHR$(255)¶
PUT #1,1¶
nr=1:GOSUB AuxOutput¶
MiniFile=1¶
LOCATE 19,12:PRINT SPACE$(70)¶
LOCATE 19,12:PRINT ActualMiniFile$¶
GOSUB MenuOut¶
GOTO CreateMask¶
¶
RecordChange:¶
Enter$=""¶
FOR i=1 TO quantity¶
xp%=1+i¶
text$=Enter$(i)¶
Buffer¶
TextDataEntry 28,xp%,80,32,text$¶
Enter$(i)=text$¶
LOCATE 1+i,28:PRINT SPACE$(32)¶
LOCATE 1+i,28¶
lang=LEN(text$):IF lang>32 THEN lang=32¶
PRINT MID$(text$,1,lang)¶
Enter$=Enter$+text$+CHR$(3)¶
NEXT i¶
GOSUB MenuOn¶
l$=STR$(LEN(Enter$))¶
LSET a$=l$¶
LSET b$=Enter$¶
PUT #1,nr¶
GOSUB MenuOut¶
GOTO MainLoop¶
¶
```

```
PrintRecord:¶
GOSUB MenuOn¶
OPEN "PRT:" FOR OUTPUT AS #2¶
FOR i=1 TO quantity¶
PRINT #2,Enter$(i)¶
NEXT i¶
FOR i=1 TO 2:PRINT #2,CHR$(10):NEXT i¶
CLOSE #2¶
GOSUB MenuOut¶
IF under=1 THEN RETURN¶
GOTO MainLoop¶
¶
MiniFilePrint:¶
GOSUB MenuOn¶
nr=1¶
OPEN "PRT:" FOR OUTPUT AS #2¶
more10:¶
ReturnChk=1:GOTO Separate2¶
R1:¶
FOR i=1 TO quantity¶
PRINT #2,Enter$(i)¶
NEXT i¶
FOR i=1 TO 2:PRINT #2,CHR$(10):NEXT i¶
nr=nr+1¶
GOTO more10¶
¶
MiniDataQuit:¶
COLOR 1,0¶
LOCATE 22,21¶
PRINT "Are you sure ? (if yes then 'y')"¶
w:¶
be$=INKEY$¶
IF be$="" THEN GOTO w¶
IF UCASE$(be$)="Y" THEN CLOSE #1:END¶
LOCATE 22,5¶
PRINT SPACE$(70)¶
GOTO MainLoop¶
¶
FirstRecord:¶
nr=1:halt=0¶
¶
apart:¶
GOSUB MenuOn¶
GOTO Separate2¶
R3:¶
FOR i=1 TO quantity¶
LOCATE i+1,28¶
PRINT Enter$(i)¶
NEXT i¶
GOSUB AuxOutput¶
GOSUB MenuOut¶
IF under=1 THEN RETURN¶
GOTO MainLoop¶
¶
PrevRecord:¶
halt=0¶
```

```
nr=nr-1:IF nr<1 THEN nr=1:GOTO MainLoop¶
GOTO apart¶
¶
NextRecord:¶
IF halt=1 THEN MainLoop¶
vor=1¶
nr=nr+1¶
GOTO apart¶
¶
DataEntry:¶
IF Enter$<>"" THEN RecordChange¶
FOR i= 1 TO quantity¶
text$=""¶
xp%=1+i¶
Buffer¶
TextDataEntry 28,xp%,80,32,text$¶
Enter$(i)=text$¶
LOCATE 1+i,28:PRINT SPACE$(32)¶
LOCATE 1+i,28¶
lang=LEN(text$):IF lang>32 THEN lang=32¶
PRINT MID$(text$,1,lang)¶
Enter$=Enter$+text$+CHR$(3)¶
NEXT i¶
GOSUB MenuOn¶
l$=STR$(LEN(Enter$))¶
LSET a$=l$¶
LSET b$=Enter$¶
PUT #1,nr¶
FIELD #1,10 AS init1$,720 AS init2$¶
LSET init1$=CHR$(1)¶
LSET init2$=CHR$(255)¶
PUT #1,nr+1¶
GOSUB MenuOut¶
halt=0¶
nr=nr+1¶
GOTO apart¶
¶
Searcher:¶
FOR i=1 TO quantity¶
LOCATE 1+i,28¶
PRINT Search$(i)¶
NEXT i¶
LOCATE 22,19:PRINT "Please input or change search
critera."¶
FOR i=1 TO quantity¶
text$=Search$(i)¶
xp%=1+i¶
Buffer¶
TextDataEntry 28,xp%,80,32,text$¶
Search$(i)=text$¶
LOCATE 1+i,28:PRINT SPACE$(32)¶
LOCATE 1+i,28¶
lang=LEN(text$):IF lang>32 THEN lang=32¶
PRINT MID$(text$,1,lang)¶
NEXT i¶
LOCATE 22,5:PRINT SPACE$(70)¶
```

```
FOR i=1 TO quantity¶
IF Search$(i)<>"" THEN start¶
NEXT i¶
LOCATE 22,24¶
PRINT "No search critera available."¶
GOSUB Pause¶
GOSUB AuxOutput¶
under=1:GOSUB apart:under=0:GOTO MainLoop¶
¶
start:¶
nr=1¶
LOCATE 22,5:PRINT SPACE$(70)¶
start2:¶
GOSUB MenuOn¶
ReturnChk=2:GOTO Separate2¶
R2:¶
FOR i=1 TO quantity¶
IF Search$(i)<>"" AND INSTR(Enter$(i),Search$(i))=0 THEN
moreIV¶
NEXT i¶
FOR i=1 TO quantity¶
LOCATE 1+i,28¶
PRINT Enter$(i)¶
NEXT i¶
GOSUB MenuOut¶
LOCATE 22,17:PRINT "F1=Print Record      Key=Search ..."¶
question:¶
ab$=INKEY$¶
IF ab$="" THEN question¶
IF ab$=CHR$(129) THEN under=1:GOSUB PrintRecord:under=0¶
LOCATE 22,5:PRINT SPACE$(70)¶
moreIV:¶
nr=nr+1¶
GOTO start2¶
¶
MaskLoad:¶
COLOR 2,0¶
OPEN MiniFileName$ FOR INPUT AS #3¶
INPUT #3,quantity¶
FOR i=1 TO quantity¶
INPUT #3,Maske$(i)¶
LOCATE i+1,2:PRINT "(";i:LOCATE i+1,5:PRINT ")"¶
LOCATE i+1,7¶
PRINT Maske$(i)¶
NEXT i¶
CLOSE #3¶
COLOR 1,0¶
RETURN¶
 ¶
MaskeSave:¶
MiniFileName$=""¶
GOSUB Separate¶
GOSUB MenuOn¶
OPEN MiniFileName$ FOR OUTPUT AS #3¶
PRINT #3,quantity¶
FOR i=1 TO quantity¶
```

```
PRINT #3,Maske$(i)¶
NEXT i¶
CLOSE #3¶
GOSUB MenuOut¶
IF under=1 THEN RETURN¶
GOTO MainLoop¶
¶
CreateMask:¶
FOR i=1 TO quantity¶
Maske$(i)=""¶
LOCATE 1+i,20¶
PRINT SPACE$(17)¶
NEXT i¶
again:¶
LOCATE 22,5¶
PRINT "Number of Fields per Record (max. 9) :"¶
quantity=0:IF other=0 THEN text$=""¶
Buffer¶
TextDataEntry 46,22,1,2,text$¶
LOCATE 22,5:PRINT SPACE$(70)¶
quantity=VAL(text$)¶
IF quantity<1 OR quantity>9 THEN again¶
mcreateII:¶
FOR i=1 TO quantity¶
 text$=""¶
REM IF other=1 THEN text$=Maske$(i)¶
¶
¶
xp%=1+i¶
COLOR 2,0¶
LOCATE xp%,2:PRINT "(";i:LOCATE xp%,5:PRINT ")"¶
Buffer¶
TextDataEntry 7,xp%,19,20,text$¶
IF RIGHT$(text$,1)<>" " AND RIGHT$(text$,1)<>"." THEN
text$=text$+" "¶
lang:¶
IF LEN(text$)<20 THEN text$=text$+".":GOTO lang¶
COLOR 1,0:LOCATE 1+i,7:PRINT SPACE$(18)¶
COLOR 2,0:LOCATE 1+i,7:PRINT text$¶
Maske$(i)=text$¶
NEXT i¶
other=0¶
COLOR 1,0¶
GOTO MaskeSave¶
¶
MaskChange:¶
other=1¶
GOTO mcreateII¶
¶
Pause:¶
FOR i=1 TO 4:MENU i,0,0:NEXT i¶
Buffer¶
LOCATE 22,63¶
PRINT "ª Press a key ´"¶
WHILE INKEY$="":WEND¶
LOCATE 22,5:PRINT SPACE$(70)¶
```

```
LOCATE 22,63:PRINT SPACE$(16)¶
FOR i=1 TO 4:MENU i,0,1:NEXT i¶
RETURN¶
¶
AuxOutput:¶
COLOR 1,0¶
LOCATE 20,17:PRINT SPACE$(10)¶
LOCATE 20,17:PRINT STR$(nr)¶
RETURN¶
¶
Interruption:¶
CLOSE #1:END¶
¶
Problem:¶
GOSUB MenuOut¶
COLOR 1,0¶
LOCATE 22,5:PRINT SPACE$(70)¶
IF ERR=7 OR ERR=14 THEN¶
  LOCATE 22,18:PRINT "Memory full."¶
  RESUME Marke¶
END IF¶
IF ERR=53 THEN¶
  LOCATE 22,19:PRINT "File not found."¶
  LOCATE 19,12:PRINT SPACE$(63)¶
¶
  CLOSE #1¶
  KILL ActualMiniFile$¶
  MiniFile=0¶
  RESUME Marke¶
END IF¶
LOCATE 22,17¶
PRINT "An Internal program error occoured."¶
Marke:¶
GOSUB Pause¶
GOTO MainLoop¶
¶
Separate:¶
MiniFileName$=""¶
FOR i=1 TO LEN(ActualMiniFile$)¶
IF MID$(ActualMiniFile$,i,1)="." THEN stop1¶
MiniFileName$=MiniFileName$+ MID$(ActualMiniFile$,i,1)¶
NEXT i¶
stop1:¶
MiniFileName$=MiniFileName$+".Maske"¶
RETURN¶
¶
Separate2:¶
z=0:n=1:Enter$=""¶
GET #1,nr¶
l$=a$:Enter$=b$¶
IF INSTR(Enter$,CHR$(255))<>0 THEN¶
  GOSUB MenuOut¶
  IF ReturnChk=1 THEN CLOSE #2:ReturnChk=0:GOTO
FirstRecord¶
  GOSUB AuxOutput¶
  under=0¶
```

```
  IF ReturnChk=2 THEN¶
    LOCATE 22,21:PRINT "No more records available."¶
    ReturnChk=0¶
    GOSUB Pause¶
    GOTO FirstRecord¶
  END IF¶
  FOR i=1 TO quantity:Enter$(i)="":NEXT i:Enter$=""¶
  halt=1¶
END IF¶
l=VAL(l$)¶
FOR i=1 TO l¶
IF MID$(Enter$,i,1)=CHR$(3) THEN¶
  z=z+1:IF z>quantity THEN ende¶
  Enter$(z)=MID$(Enter$,n,i-n)¶
  n=i+1¶
END IF¶
NEXT i¶
ende:¶
IF ReturnChk<>0 THEN ON ReturnChk GOTO R1,R2¶
GOTO R3¶
¶
MenuOn:¶
FOR i=1 TO 5:MENU i,0,0:NEXT i¶
LOCATE 22,62:PRINT "ª Moment ... ´"¶
RETURN¶
¶
MenuOut:¶
FOR i=1 TO 5:MENU i,0,1:NEXT i¶
LOCATE 22,62:PRINT SPACE$(14)¶
Buffer¶
RETURN¶
¶
SortRoutine:¶
text$=""¶
LOCATE 22,5¶
PRINT "Sort using which field :"¶
Buffer¶
TextDataEntry 32,22,2,3,text$¶
LOCATE 22,5:PRINT SPACE$(70)¶
IF text$="" OR VAL(text$)<1 OR VAL(text$)>quantity THEN
MainLoop¶
Kriterium=VAL(text$)¶
GOSUB MenuOn¶
nr=1¶
more:¶
z=0:n=1:Enter$=""¶
GET #1,nr¶
l$=a$:Enter$=b$¶
IF INSTR(Enter$,CHR$(255))<>0 THEN more2¶
nr=nr+1¶
GOTO more¶
¶
more2:¶
Counter=nr-1¶
DIM DataEntry2$(Counter)¶
FOR k=1 TO Counter¶
```

```
z=0:n=1:Enter$=""¶
GET #1,1¶
l$=a$:Enter$=b$¶
l=VAL(l$)¶
FOR j=1 TO l¶
IF MID$(Enter$,j,1)=CHR$(3) THEN¶
  z=z+1:IF z>quantity THEN ende2¶
  Enter$(z)=MID$(Enter$,n,j-n)¶
  n=j+1¶
END IF¶
NEXT j¶
ende2:¶
FOR i=1 TO Counter-1¶
z=0:n=1:DataEntry2$=""¶
GET #1,i+1¶
l2$=a$:DataEntry2$=b$¶
l2=VAL(l2$)¶
FOR j=1 TO l2¶
IF MID$(DataEntry2$,j,1)=CHR$(3) THEN¶
  z=z+1:IF z>quantity THEN ende3¶
  DataEntry2$(z)=MID$(DataEntry2$,n,j-n)¶
  n=j+1¶
END IF¶
NEXT j¶
ende3:¶
IF Enter$(Kriterium) > DataEntry2$(Kriterium) THEN¶
  LSET a$=l$¶
  LSET b$=Enter$¶
  PUT #1,i+1¶
  LSET a$=l2$¶
  LSET b$=DataEntry2$¶
  PUT #1,i¶
  GOTO iandk¶
END IF¶
Enter$=DataEntry2$:l$=l2$¶
FOR a=1 TO quantity:Enter$(a)=DataEntry2$(a):NEXT a¶
iandk:¶
NEXT i¶
NEXT k¶
ERASE DataEntry2$¶
GOSUB MenuOut¶
GOTO FirstRecord¶
¶
SUB Buffer STATIC¶
Buffer:¶
ad$=INKEY$¶
IF ad$<>"" THEN ad$="":GOTO Buffer¶
END SUB¶
¶
SUB TextDataEntry (xpos%,ypos%,Length%,Wide%,text2$)
STATIC¶
SHARED text$¶
text$=text2$¶
COLOR 0,2¶
LOCATE ypos%,xpos%:PRINT SPACE$(Wide%)¶
COLOR 1,2¶
```

```
IF text$<>"" THEN LOCATE ypos%,xpos%:PRINT text$¶
quantity=0:StepNum=1:xpos2%=xpos%¶
LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
¶
1 :¶
ab$=INKEY$¶
IF ab$="" THEN 1¶
IF ab$=CHR$(3) OR ab$=CHR$(255) THEN 1¶
  ¶
'Ende                    ¶
IF ab$=CHR$(13) THEN goback10¶
¶
'Cursor right¶
IF ab$=CHR$(30) AND text$<>"" AND quantity<LEN(text$)
THEN¶
  LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),2  ¶
  IF StepNum>0 THEN LOCATE ypos%,xpos2%:PRINT
MID$(text$,StepNum,1)¶
  xpos%=xpos%+1¶
  IF xpos%>xpos2%+Wide%-1 THEN¶
    xpos%=xpos2%+Wide%-1¶
    StepNum=StepNum+1¶
    IF (StepNum-1)>50 THEN StepNum=50¶
  END IF¶
lang=LEN(text$):IF lang>Wide% THEN lang=Wide%  ¶
IF StepNum>0 THEN LOCATE ypos%,xpos2%:PRINT
MID$(text$,StepNum,lang)¶
LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
quantity=quantity+1¶
GOTO 1¶
END IF¶
IF ab$=CHR$(30) THEN 1¶
¶
'Cursor left¶
IF ab$=CHR$(31) AND text$<>"" AND quantity>0 THEN¶
  LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),2¶
  IF StepNum>0 THEN LOCATE ypos%,xpos2%:PRINT
MID$(text$,StepNum,1)¶
  xpos%=xpos%-1¶
  IF xpos%<xpos2% THEN¶
    xpos%=xpos2%¶
    StepNum=StepNum-1¶
    IF (StepNum-1)<1 THEN StepNum=1¶
  END IF¶
lang=LEN(text$):IF lang>Wide% THEN lang=Wide%¶
IF StepNum>0 THEN LOCATE ypos%,xpos2%:PRINT
MID$(text$,StepNum,lang)¶
LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
quantity=quantity-1¶
'GOTO 1¶
END IF¶
IF ab$=CHR$(31) THEN 1¶
¶
'Backspace¶
IF ab$=CHR$(8) AND quantity>0 AND text$<>"" THEN¶
```

```
  text$=LEFT$(text$,quantity-
1)+MID$(text$,quantity+1,LEN(text$)-quantity)¶
  xpos%=xpos%-1:quantity=quantity-1¶
  lang=LEN(text$):IF lang>Wide% THEN lang=Wide%  ¶
  LINE (xpos%*8-8,ypos%*8-8)-((Wide%+xpos2%-1)*8-
1,ypos%*8-1),2,bf¶
  LOCATE ypos%,xpos2%:PRINT MID$(text$,StepNum,lang)¶
  LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
  GOTO 1¶
END IF¶
IF ab$=CHR$(8) THEN 1¶
¶
' Delete ¶
IF ab$=CHR$(127) AND quantity>=0 AND text$<>"" THEN¶

text$=LEFT$(text$,quantity)+MID$(text$,quantity+2,LEN(tex
t$)-quantity)¶
  lang=LEN(text$):IF lang>Wide% THEN lang=Wide%¶
  LINE (xpos%*8-8,ypos%*8-8)-((Wide%+xpos2%-1)*8-
1,ypos%*8-1),2,bf¶
  LOCATE ypos%,xpos2%:PRINT MID$(text$,StepNum,lang)¶
  LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
  GOTO 1¶
END IF¶
IF ab$=CHR$(127) THEN 1¶
¶
¶
'DataEntry¶
IF LEN(text$)+1>Length% THEN 1¶
IF LEN(text$)>1 AND MID$(text$,quantity+1)<>"" THEN
text$=LEFT$(text$,quantity)+ab$+MID$(text$,quantity+1,LEN
(text$)-quantity) ELSE text$=text$+ab$¶
quantity=quantity+1¶
xpos%=xpos%+1¶
IF xpos%>xpos2%+Wide%-1 THEN ¶
  xpos%=xpos2%+Wide%-1¶
  StepNum=StepNum+1     ¶
  lang=LEN(text$):IF lang>Wide% THEN lang=Wide%¶
  LOCATE ypos%,xpos2%:PRINT MID$(text$,StepNum,lang)¶
  LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
  GOTO 1¶
END IF¶
lang=LEN(text$):IF lang>Wide% THEN lang=Wide%¶
LOCATE ypos%,xpos2%:PRINT MID$(text$,StepNum,lang)¶
LINE (xpos%*8-8,ypos%*8-1)-(xpos%*8-1,ypos%*8-1),3¶
GOTO 1¶
         ¶
goback10:¶
COLOR 1,0¶
END SUB
```

The ¶ character in the previous program lines is only for reference. It shows the end of the AmigaBasic line. Due to the formatting of this book many of the lines have been split. Following are the most important variables and their meaning:

Variable	Function
Enter$()	Stores data fields.
Enter$	
l$	
Makse$()	Contains template.
Search$()	Contains search criteria file
Minifile=1	file often.
ActualMiniFile$	Name of the opened file.
other	Flag if data set was already written.
nr	Number of the current data set.

Those who are interested in the labels for the example subroutines will find the labels briefly explained here:

Label	Function
MainLoop	Main program loop.
Menuon	Deactivates menu bar.
MenuOff	Activates menu bar.
TextDataEntry	Line editor for input.
Buffer	Prevents keyboard overrun.
CreateMask	Creates screen template.
NewFile	Initializes new file.
Mask change	Change screen template.
RecordChange	Change data record.

4.3 Instructions for Mini Data

Now that the theoretical part is over let's get practical. Enter or load the program from the optional disk and type RUN.

After a brief initialization period the title display appears. It's still rather empty. Only File: and Record: are displayed, but these can be changed immediately by loading an address file.

Even though Mini Data can be used from the menu bar and the keyboard, for the sake of simplicity it is used here only with the keyboard. The menu titles are self explanatory.

First select the New File item (new file) by pressing F2 and enter the name of the desired file. How about Addresses? OK, the name is input and the filename indicated appears beside File:. After a brief wait another prompt appears. This time the input refers to the number of data fields within each record of the file. Seven data fields for an address file should be enough. Therefore input 7 and press the <Return> key. This time no wait is required. The Data field template can be input immediately. The Line text editor operates using the following keys: <Delete>, <Backspace>, <Cursor up> and <Cursor down>. After pressing the Return key, the cursor jumps down and forward to the next field. The input is ended on the last, i.e. lowest, data field template, the template is stored automatically. With this basic knowledge the following input template can be created:

```
Name ...............
Address ............
City ...............
State ..............
Zip ................
Telephone ..........
Remarks ............
```

The dots behind the names of the data fields are added by Mini Data. Of course the templates and inputs suggested here are not compulsory, but it is easier to follow the example if we use the same data.

After installing the template the input can start. Simply press the <Return> key. Mini Data is now ready for input of data and signals this by a highlighted text field and a red cursor. In the input mode, every text field has a preset maximum of 80 characters. Since they do not all fit into the text window (in Mini Data a text window can hold only 32 characters), the text scrolls in the text window. Very practical! Try the first input:

```
Name ............... Jim Smith          <Return>
Address ............ 1234 Any drive     <Return>
City ............... Somewhere          <Return>
State .............. MI                 <Return>
Zip ................ 49505              <Return>
Telephone .......... xxx-xxx-xxxx       <Return>
Remarks ............ none               <Return>
```

The <Returns> behind the input lines should not be input, but simply mean to press the <Return> key. After the input has been completed, the disk drive runs briefly. On the lower right side of the screen a message appears requesting you to wait a moment. During this time no other function can be selected to prevent disturbing the drive during its work. In the meantime the message "Record" has changed to indicate the second data record even though only one has been input. Simple, if an entry was found in a previously unused data record, it is stored and a switch occurs at the same time to the next data record to make fluid input of data possible. If a previous data record is made available for change by the pressing of the <Return> key, no switching occurs. Since the first data record was input, it was stored and after pressing the <Return> key, input of data can proceed. Input the names of a few more friends so that you can practice using Mini Data.

Next let's examine the function of the cursor keys more closely.

```
                   first data record
Previous data record ⇐ ⇑  ⇒ next data record
```

Pressing the <Cursor up> key displays the first data record of the current file. The <Cursor left> and <right> keys switch to the previous or next record. The <Cursor down> key has no significance. These three cursor keys are well suited for "paging" through a file.

Sorting files

Interested in sorting the file alphabetically? Simply press the F8 key and immediately a prompt appears which asks according to which criteria. The number which is in front of the desired template field should be input. Entering the number and pressing the <Return> key sets the entire process in motion. This function requires at least 40 seconds (to infinity, depending on the file length). This is dependent on the disk-oriented processing of Mini Data, but assures 100% data security during a system crash (which hopefully will never occur). The drive should stop after sorting and the first data set of the newly sorted file is displayed.

Nearly all functions of Mini Data have been explained, except searching and the two print options "print record" and "print file", which use the printer settings in Preferences and are self explanatory. The search routine requires you to enter the search critera and then the file is searched for all matches. A program which is easy to learn and which is useful. What more can you ask for?

4.4 AmigaBASIC improvements

Since we have discussed some material that was rather hard to digest, two special delicacies will be served for the final portion of this BASIC section. First: access to the CLI from AmigaBASIC and second: reading the disk directory. The Merge example which we promised you earlier completes the chapter.

The listing of Program 4 which permits direct access to the CLI from BASIC starts the section:

Program 4:

```
'Program 4:
DECLARE FUNCTION xOpen& LIBRARY
DECLARE FUNCTION Execute& LIBRARY

LIBRARY "dos.library"

NewCommand:
INPUT "1>";Command$
Reaction:
Command$=Command$+CHR$(0)
Display$="CON:0/0/640/200/CLI-Basic"+CHR$(0)
REM PAL screens can use
REM Display$="CON:0/0/640/256/CLI-Basic"+CHR$(0)'
connection&=xOpen&(SADD(Display$),1006)
extra&=Execute&(SADD(Command$),0,connection&)
FOR i=1 TO 20000:NEXT i
CALL xClose(connection&)

INPUT "more (y) ";w$
IF w$="y" THEN NewCommand
LIBRARY CLOSE
END
```

Those who have examined AmigaBASIC more closely are familiar with the Library command which permits use of the Amiga System libraries. This command is in this program so the dos.bmap file must be accessible on disk.

Certain expressions are declared as functions (DECLARE FUNCTION). The DOS operating system library is then opened with "dos.library" which controls among other things, the processing of CLI commands. After a brief interrogation of the desired CLI command (in Command$) the actual processing procedure occurs. To mark the end of the command sequence which was input, a CHR$(0) is attached to the variable. Next parameters are passed which open the CLI window. These have a fixed format just like most system routine accesses from BASIC. The values for width, height and the name of the window can be changed at will in the Display$ variable.

The heart of the program is in the following line:

```
extra&=Execute&(SADD(Command$),0,connection&)
```

The CLI command which was input previously is passed directly to the DOS library for processing. When the process is finished, the library is closed again (LIBRARY CLOSE).

Now some practical applications. A few examples are presented which demonstrate what can be done with CLI BASIC. For example "dir" can be used to display the directory of a disk (how dull). But wait, there's more! Basically nearly all CLI commands can be accessed from the CLI user interface. Commands which are not directly addressable are those which require an input from the user. A good example is the Setdate command. Programs cannot be started with RUN because this confuses the Amiga and sometimes causes a "Guru meditation."

An interesting example is opening a new CLI window with Newcli in which all commands can be executed correctly. It's now possible to start programs which are independent of AmigaBASIC. However, control through AmigaBASIC is no longer possible and return to the normal BASIC level is only possible with an EndCLI. In this manner work with the CLI and AmigaBASIC can be done in parallel.

The Info command should not be forgotten. It provides a complete overview of the disk. With the List command more detailed information can be obtained such as the length of the files.

The pseudo CLI can be used for many things, from setting the system time (NewCLI-Setdate-EndCLI), to copying of titles or erasing them (copy/delete), installing a RAM disk (dir ram:) or displaying the directory of a disk (dir df0:) quickly.

Bear in mind the following: The current directory must contain the dos.bmap file and the CLI commands, such as dir, NewCLI, EndCLI, Delete, Copy, etc. must be located in the C directory.

4.5 Reading a directory from BASIC

The directory of any disk can be read directly from BASIC with this program.

Program 5:

```
'Program 5:
DECLARE FUNCTION Examine& LIBRARY
DECLARE FUNCTION ExNext& LIBRARY
DECLARE FUNCTION Lock& LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION IoErr& LIBRARY

LIBRARY "exec.library"
LIBRARY "dos.library"

more2:
INPUT "Directory ";Dir$

Hello%=-2
Dir$=Dir$+CHR$(0)
bytes&=252
lock2&=Lock&(SADD(Dir$),Hello%)
opt&=2^1+2^16
info&=AllocMem&(bytes&,opt&)
suc&=Examine&(lock2&,info&)

more:
DirName&=info&+8
FOR search=0 TO 29
check=PEEK(DirName&+search)
IF check<>0 THEN
  check$=check$+CHR$(check)
ELSE
  search=29
END IF
NEXT search

DirName$=check$:check$=""
prot&=PEEKL(info&+116)

IF prot&<>0 THEN
  IF (prot& AND 2^3)<>0 THEN prot$=prot$+"read "
  IF (prot& AND 2^2)<>0 THEN prot$=prot$+"write "
  IF (prot& AND 2^1)<>0 THEN prot$=prot$+"Execute "
  IF (prot& AND 2^0)<>0 THEN prot$=prot$+"erase "
  DirProt$=LEFT$(prot$,LEN(prot$)-1)
  prot$="d"
END IF
```

```
type&=PEEKL(info&+120)
IF type&<0 THEN
  DirType$="File"
ELSEIF counter%=0 THEN
  DirType$="Directory"
ELSE
  DirType$="Directory"
END IF

DirSize&=PEEKL(info&+124)
DirBlks&=PEEKL(info&+128)

FOR search=0 TO 79
check=PEEK(info&+144+search)
IF check<>0 THEN
  check$=check$+CHR$(check)
ELSE
  search=79
END IF
NEXT search

DirComm$=check$:check$=""
suc&=ExNext&(lock2&,info&)
IF suc&=0 THEN CLS:GOTO more2

CLS
LOCATE 5,3

COLOR 3:PRINT DirName$;:COLOR 1
PRINT " is a ";
COLOR 3:PRINT DirType$;:COLOR 1:PRINT "."
IF DirType$="Directory" THEN pause
PRINT " Following Protect-Options are used:"
PRINT:COLOR 2:PRINT "  ";DirProt$:COLOR 1
pause:
PRINT :PRINT "  Continue => Key          New Dir => q"
pause2:
a$=INKEY$:IF a$="" THEN pause2
IF a$="q" THEN CLS:GOTO more2
GOTO more
```

As in the Pseudo `CLI`, the Library command is used here to access the operating system routines. The functions of the program are easily explained. After the start the directory to be listed (for example df0:) is read. The File type and the Protect mode of the files are output. The exe.bmap and dos.bmap files must be in the current directory.

4.6 MERGE

The capabilities of the MERGE command will be explained in this section. A simple example program has been selected for this task to illustrate MERGE clearly. The MERGE command does not occur in the program itself, it would not be efficient in a program of this size. First let's examine the listing:

Program 6:

```
'Program 6:
'MiniBase V1.0

REM ON ERROR GOTO Problem
SCREEN 1,320,200,4,1
WINDOW 1,"Mini Base V1.0",,0,1

DIM Entry(12)
PALETTE 0,0,0,0

PALETTE 1,0,0,0
PALETTE 2,1,1,1
COLOR 2,0

INPUT "Data is (D)Data  Statements or (I)Input ";a$
IF UCASE$(a$)="Q" THEN Ende
IF UCASE$(a$)="D" THEN
  FOR i=1 TO 12
  READ Entry(i)
  NEXT i
  GOTO BarChart
END IF

DataEntry:
CLS
PRINT "Input :"
PRINT
FOR i=1 TO 12
RepeatEntry:
PRINT "Value Nr.";i;
INPUT Entry(i)
IF Entry(i)=-1 THEN Ende
IF Entry(i)<0 OR Entry(i)>20 THEN PRINT "False input;
repeat...":GOTO RepeatEntry


NEXT i

BarChart:
CLS
FOR i=1 TO 12
COLOR 1,3+i
```

```
FOR r=1 TO Entry(i)
LOCATE 23-r,i*3:PRINT "  "
NEXT r
COLOR 2,0:LOCATE 23-r-1,i*3-1:PRINT Entry(i)
NEXT i

a$=""
LOCATE 23,9:INPUT "(S)ave or (N)ew ";a$
IF UCASE$(a$)="Q" THEN Ende
IF UCASE$(a$)="S" THEN
  REM LOCATE 23,9:PRINT SPACE$(16);
  LOCATE 23,9:INPUT "Filename         ";file$
  OPEN file$ FOR OUTPUT AS #1
  PRINT #1,"EnteredData:";CHR$(13)
  FOR i=1 TO 12
  PRINT #1,"DATA ";Entry(i);CHR$(13)
  NEXT i
  CLOSE #1
END IF

GOTO DataEntry

Problem:
IF ERR=4 THEN
  CLS
  PRINT "No Data available !!!  [Key]"
  WHILE INKEY$="":WEND
  ON ERROR GOTO ERROR
  RESUME DataEntry
END IF
END

Ende:
WINDOW CLOSE 1
SCREEN CLOSE 1
END
```

To relieve the user of a typing chore, this program is also included in the BASIC drawer of the optional disk for this book.

A description

MiniBase is a small graphic calculation program, which displays the values input as a bar graph on the screen and saves them.

After starting the program a prompt asks whether the data exists as data lines at the end of the program or if it should be input. Now either press the <Return> key or press <I> <Return>.

The number of data has been set at 12 data items. If less are input, the prompt can be answered with a Return. To change the number of items possible do the following: Every place where the number 12 occurs, substitute the number of items you want to input.

The maximum size of the data to be input is 20. Only whole numbers (without a decimal point) can be used. When the last number has been input, the bar chart with its values is constructed on the screen. You are then asked if the data should be stored as a file, or if new data is input. The new input erases the old data. An <s> is input for saving the data. After the drive has finished running, the program returns to Input mode. To terminate the program, press <Ctrl><C>, input -1 in Input mode or a <q> for Quit during all other prompts.

The MERGE command can be used to append the saved data to the program.

5.
AmigaDOS

5. AmigaDOS

The operating system of the Amiga is partitioned into various hierarchical levels. The lowest level is the individual device drivers, for example the disk drive, the serial interface, the keyboard and the screen (Console). This low level offers only limited capabilities. Normally the transfer rate is in 1K increments from the computer to a device and vice versa.

No user will accept this slow rate. A disk directory at this rate would take hours! AmigaDOS is responsible for the upper level of the operating system. All threads of the individual devices come together here. For example, it is possible to redirect a task which normally would have been displayed on screen to the printer or a file.

AmigaDOS cannot only handle the devices, but it also controls the `CLI`. This part of AmigaDOS is less interesting for the user since this book deals with the disk. But even in this area AmigaDOS can offer a few features.

5.1 BCPL-variables under AmigaDOS

AmigaDOS, like most of the system software of the Amiga, was developed in BCPL. BCPL is a predecessor of the widely available C programming language and has some peculiarities which must be observed while programming in C.

The user who has looked at the Include files of the C compiler may have already noticed some peculiar variable types. These are the variables of BCPL which were translated into C:

```
BPTR = BCPL-Pointer
BSTR = BCPL-String
```

The BPTR is a pointer into the memory of the Amiga just like a C pointer. The BCPL pointer only points to memory addresses which are divisible by 4. It counts in long words (32 bits) instead of bytes (8 bit). In the real world the conversion from C pointers (for example APTR) to BCPL appears as follows:

```
BCPL = APTR / 4 (APTR must be completely divisible by 4!)
APTR = BCPL * 4
```

The Include file "libraries/dos.h" contains a helpful conversion routine. It shifts the bits of the BPTR to the left by two and thus multiplies it by 4.

```
typedef long BPTR;
typedef long BSTR;
#define BADDR( bptr ) (((ULONG)bptr)<<2)
Usage: APTR = BADDR( BPTR )
```

The BCPL string BSTR works just like the BPTR in the long word format. It is a pointer to a series of bytes which contain character codes. Unlike C, the first byte of the BCPL string contains the length of the character string instead of the first character. This is followed by the actual characters.

5.2 Internal organization of AmigaDOS

AmigaDOS is a library, like all parts of the Amiga operating system. However for C programs AmigaDOS has a special library. It doesn't have to be opened like the Intuition library before using it in programs. The opening is performed by the initialization routine which is automatically linked to each C program. The base pointer of the library is then stored in the global variable DOSBase.

Below the DOSBase address (with smaller addresses) are the addresses of the individual DOS routines as in any library. Starting with positive offsets from the DOSBase (addresses larger than the DOSBase) the data area of the DOS library can be reached. This is a structure containing the pointers to all additional internal data of AmigaDOS:

```
struct DosLibrary {
struct     Library   dl_lib;
           APTR      dl_Root;
           APTR      dl_GV;
           LONG      dl_A2;
           LONG      dl_A5;
           LONG      dl_A6;
};
```

The only interesting entry is "dl_Root". It points to another structure, the "RootNode":

```
struct RootNode {
           BPTR      rn_TaskArray;
           BPTR      rn_ConsoleSegment;
struct     DateStamp rn_Time;
           LONG      rn_RestartSeg;
           BPTR      rn_Info;
           BPTR      rn_FileHandlerSegment;
};
```

Most of the entries of this structure are used for tasks which AmigaDOS performs outside the I/O control (`CLI` etc.).

The pointer in "rn_Info" is important. It points to the DOS Info structure which has the following structure:

```
struct DosInfo {
           BPTR      di_McName;
           BPTR      di_DevInfo;
           BPTR      di_Devices;
           BPTR      di_Handlers;
           APTR      di_NetHand;
};
```

This structure is the key to all devices known to AmigaDOS. In Version 1.2 of AmigaDOS only the di_DevInfo entry is occupied. Here we find another pointer which points to another structure.

```
struct DeviceNode {
          BPTR      dn_Next;
          ULONG     dn_Type;
struct    MsgPort  *dn_Task;
          BPTR      dn_Lock;
          BSTR      dn_Handler;
          ULONG     dn_StackSize;
          LONG      dn_Priority;
          BPTR      dn_Startup;
          BPTR      dn_SegList;
          BPTR      dn_GlobalVec;
          BSTR      dn_Name;
};
```

This type of structure exists for every mounted device (for example PAR for the parallel interface). The first entry "dn_Next" always points to the next DeviceNode structure. The last structure which cannot point to another one, contains a null.

AmigaDOS controls not only the individual devices with this structure. Disks (volumes) and directories can be declared as pseudo devices in this manner. The CLI command Assign can use this list to make the C directory of the Workbench disk into a logical device.

The device in question is listed in "dn_Type". It can assume the following values:

```
#define DLT_DEVICE       0L
#define DLT_DIRECTORY    1L
#define DLT_VOLUME       2L
```

For the type "DTL_VOLUME" there is another modified form of the DeviceNode structure:

```
struct DeviceList {
          BPTR      dl_Next;
          LONG      dl_Type;
struct    MsgPort  *dl_Task;
          BPTR      dl_Lock;

struct    DateStamp dl_VolumeDate;
          BPTR      dl_LockList;
          LONG      dl_DiskType;
          LONG      dl_unused;
          BSTR     *dl_Name;
};
```

Of interest here is the dl_Lock or the dn_Lock entry. It again points to an entire list of structures. For every file of a disk one Lock structure can exist.

One function of Lock is to prevent, for example, two parallel executing tasks writing to the same file at the same time. This would lead to chaos! If a task is currently writing to a file, a Lock structure marks it so no other task can access this file until writing has been completed. More on this in the next section.

The following example program outputs all devices which are registered in AmigaDOS. It determines the RootNode structure using the DOS base address and climbs through the previously mentioned structures to the first DeviceList. Since the structures are connected with each other through the BCPL pointer they must first be converted with BADDR() into C pointers. Then the program uses a while loop to read the pointers for all DeviceList or DeviceNode structures. The type and name for every device is output. Since the name is a BCPL string, it must be converted with "printf" into a C string before output.

```
/*----------------------------------------------------*/
/*          ASSIGN-Function for AmigaDOS              */
/*                                                    */
/*                JEA, 18-08-87                       */
/*----------------------------------------------------*/
#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include <libraries/filehandler.h>

extern struct DosLibrary *DOSBase;
UBYTE *dtl_types[] = {
"Device   : ",
"Directory: ",
"Volume   : "
};

/*----------------------------------------------------*/
/*           Convert BSTR into C-String               */
/*                                                    */
/*                                                    */
/*----------------------------------------------------*/
BstrC( bstr, buf )
BSTR *bstr;
UBYTE *buf;
{
UBYTE *str;
LONG loop;
LONG counter;

   counter = 0;
   str = (UBYTE*) BADDR( bstr );
   for( loop = (LONG) str[0]; loop--; ++counter){
      buf[counter] = str[counter+1];
   }
   buf[counter] = 0;
}

/*----------------------------------------------------*/
/*               output BPTR-String                   */
/*                                                    */
/*                                                    */
/*----------------------------------------------------*/
BstrOut( bstr )
```

```
BSTR *bstr;
{
UBYTE buf[80];

   BstrC( bstr, buf );
   printf( buf );
}

/*---------------------------------------------------*/
/*              Output ASSIGN Entries                 */
/*                                                    */
/*                                                    */
/*---------------------------------------------------*/
FindeAssign()
{
struct RootNode   *rootnode;
struct DosInfo    *dosinfo;
struct DeviceList *devicelist;
struct FileLock   *filelock;

  rootnode   = (struct RootNode*)   DOSBase->dl_Root;
  dosinfo    = (struct DosInfo*)    BADDR( rootnode->rn_Info );
  devicelist = (struct DeviceList*) BADDR( dosinfo->di_DevInfo );
   while( devicelist->dl_Next ){
      printf( dtl_types[devicelist->dl_Type] );
      BstrOut( devicelist->dl_Name );
      printf( "\n" );
devicelist = (struct DeviceList*) BADDR( devicelist->dl_Next );
   }
}

/*---------------------------------------------------*/
/*                     Main program                   */
/*                                                    */
/*                                                    */
/*---------------------------------------------------*/
main()
{
   FindeAssign();
}
```

5.3 The functions of AmigaDOS

AmigaDOS has a set of 23 routines which permit control of the devices. These file handling routines are easy to program because of their high level in the operating system. On this level, as already mentioned, all devices are treated equally. The following program reads a file and outputs it to the screen. Some DOS functions are used which are explained at the end of the program. The program shows that it is easy for DOS to address various output devices such as the drive and screen display.

```
/*-------------------------------------------------------------*/
/*              Call of the  DOS-Functions                      */
/*                                                              */
/*                     JEA, 18-08-87                            */
/*-------------------------------------------------------------*/
#include <exec/exec.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include <libraries/filehandler.h>
#include <stdio.h>

char *error_strs[] = {
"ERROR_NO_DEFAULT_DIR: 201",
"ERROR_OBJECT_IN_USE: 202",
"ERROR_OBJECT_EXISTS: 203",
"ERROR_DIR_NOT_FOUND: 204",
"ERROR_OBJECT_NOT_FOUND: 205",
"ERROR_BAD_STREAM_NAME: 206",
"ERROR_OBJECT_TOO_LARGE: 207",
"ERROR_ACTION_NOT_KNOWN: 209",
"ERROR_INVALID_COMPONENT_NAME: 210",
"ERROR_INVALID_LOCK: 211",
"ERROR_OBJECT_WRONG_TYPE: 212",
"ERROR_DISK_NOT_VALIDATED: 213",
"ERROR_DISK_WRITE_PROTECTED: 214",
"ERROR_RENAME_ACROSS_DEVICES: 215",
"ERROR_DIRECTORY_NOT_EMPTY: 216",
"ERROR_TOO_MANY_LEVELS: 217",
"ERROR_DEVICE_NOT_MOUNTED: 218",
"ERROR_SEEK_ERROR: 219",
"ERROR_COMMENT_TOO_BIG: 220",
"ERROR_DISK_FULL: 221",
"ERROR_DELETE_PROTECTED: 222",
"ERROR_WRITE_PROTECTED: 223",
"ERROR_READ_PROTECTED: 224",
"ERROR_NOT_A_DOS_DISK: 225",
"ERROR_NO_DISK: 226"
};
```

```
/*---------------------------------------------------------
/*
/*                AmigaDOS Error-Text output
/*                                                        */
/*--------------------------------------------------------*/
LONG
get_error()
{
LONG error;

   error = IoErr();

   if( error < 120 ){
      switch( error ){
      case 000:
         printf( "All OK!\n" );
         break;
      case 103:
         printf( "ERROR_NO_FREE_STORE: 103\n" );
         break;
      case 105:
         printf( "ERROR_TASK_TABLE_FULL: 105\n" );
         break;
      case 120:
         printf( "ERROR_LINE_TOO_LONG: 120\n" );
         break;
      case 121:
         printf( "ERROR_FILE_NOT_OBJECT: 121\n" );
         break;
      case 122:
         printf( "ERROR_INVALID_RESIDENT_LIBRARY: 122\n" );
         break;
      case 232:
         printf( "ERROR_NO_MORE_ENTRIES: 232\n" );
         break;
      }
   }
   else{
      printf( "%s\n", error_strs[error-201] );
   }
   return( error );
}

/*--------------------------------------------------------*/
/*                         Type                           */
/*                                                        */
/*                                                        */
/*--------------------------------------------------------*/
type( filename )
UBYTE *filename;
{
struct FileHandle *handle;
struct FileHandle *Open();
UBYTE buf;
LONG read_length;

   handle = Open( filename, MODE_OLDFILE );

   if( handle ){
      do{
         read_length = Read( handle, &buf, 1L );
         printf( "%c", buf );
```

```
      }
      while( read_length );
      Close( handle );
   }
   else{
      get_error();
   }
}

/*------------------------------------------------------------
*/
/*                              Main program
/*                                                            */
/*                                                            */
/*------------------------------------------------------------*/
main( arg_num, args )
int arg_num;
char *args[];
{
   if( arg_num > 1 )
      type( args[1] );
   else
      printf( "A file name is required!\n" );
}
```

5.4 DOS functions

In this chapter all DOS functions, offsets and the registers in which the various parameters must be passed are discussed.

5.4.1 General Input/Output functions

Open

```
Handle = Open (Name, Mode)
 D0      -30    D1   D2
```

Opens a file

Opens the file to which D1 points. Text must be terminated with a null byte.

The Mode in D2 can be Mode_readwrite (1004 for DOS 1.2) for input/output, Mode_old (1005) for input from or Mode_new (1006) for output to the file.

In D0 a pointer to the File Handle structure is returned, or a null when the function could not be performed. The File Handle structure has the following format:

Offset	Name	Significance
0	Link	Unused.
4	Interact	If <>0, the file is interactive.
8	ID	Identification number of the file.
12	Buffer	Pointer to internal storage needed.
16	CharPos	Pointer required internally.
20	BufEnd	Pointer required internally.
24	ReadFunc	Pointer to routine which is called when buffer is empty.
28	WriteFunc	Pointer to routine which is called when buffer is full.
32	CloseFunc	Pointer to routine which is called during closing of the file.
36	Argument1	
40	Argument2	File type dependent arguments.

Most entries are provided for the internal usage of AmigaDOS. These values should not be manipulated.

Close

```
Close ( Handle )
-36      D1
```

Closes a file

Closes the file which was opened with the Open command. The pointer passed in D1 is the pointer from the Open command to the File Handle structure.

Read

```
Number = Read ( Handle, Buffer, Length )
  D0     -42     D1       D2      D3
```

Reads data

Reads bytes from the file specified by the Handle up to the Length into the memory starting at address Buffer.

The value returned in D0 indicates the number of bytes actually read. If this number is 0, the end of the file was reached. If an error occurred, -1 is returned.

Write

```
Number = Write ( Handle, Buffer, Length )
  D0     -48      D1       D2      D3
```

Writes data

Writes the number of bytes as specified in Length in the file specified by Handle from memory, starting at address Buffer.

The number of bytes actually written is returned in D0. If this value is -1, an error occurred.

Seek

```
Position = Seek ( Handle, Interval, Mode )
  D0        -66    D1       D2        D3
```

Moves file pointer

This function moves the internal pointer in the file specified by Handle. The Mode determines if the value in the Interval should move the pointer relative to the beginning of the file or the end of the file. This value is calculated according to the preceding sign so that it can also be moved backwards.

The possible modes are:

OFFSET_BEGINNING -1
OFFSET_CURRENT 0
OFFSET_END 1

The return value indicates the current position of the pointer after the execution of the function. To determine the position of the pointer at the moment, the Mode relative to the position (OFFSET_CURRENT) can be set and moved by 0K: the position returned is equal to the old position.

Input

```
Handle = Input ()
  D0     -54
```

Determines standard input channel

This function determines the Handle of the channel from which the standard input can be read. If the program was called from the CLI this is the Handle of the CLI window. If the CLI command which called the program used the Data redirection capability, the handle of the channel selected is found. An example:

```
>Programname < DF0:Filename
```

Input, resulting from the Read command, to the program comes from the file named Filename.

Output

```
Handle = Output()
```

Determines standard output channel

This function determines the Handle of the channel where the standard output can be written. If the program was called from the CLI this is the Handle of the CLI window. Here also the standard output can be redirected.

```
>Programname > PRT:
```

The standard output of the calling program is sent to the printer.

WaitForChar

```
Status = WaitForChar ( Handle, Timeout )
  D0        -204          D1       D2
```

Waits for a character

This function waits the number of microseconds indicated in Timeout for the character from the channel specified in the Handle (for example the RAW: window). If during this time a character isn't received, a 0 is returned in Status, otherwise the value -1. The character can be read with the Read function.

The function is only available if the channel is an interactive channel (interactive terminal), for example a RAW: window, in which input and output can occur at the same time and data are not necessarily required immediately.

IsInteractive

```
Status = IsInteractive ( Handle )
  D0        -216            D1
```

Determines channel type

True (-1) is returned in the Status if the channel specified by the Handle is an interactive terminal which can handle input and output. Otherwise False (0) is returned.

IoErr

```
Error = IoErr ()
D0      -132
```

Determines Input/Output error

An error is reported after the call of a function by a null as the return value in D0 (usally). The exact error message can be determined by calling IoErr. D0 contains the number of the error which occurred (see the Why command of the CLI).

A listing of the error values can be found in the next section.

5.4.2 Disk operations

CreateDir

```
Lock = CreateDir ( Name )
D0     -120         D1
```

Creates a sub-directory

A sub-directory is created named Name in the current directory. The return value sets a pointer to a file structure (Lock) which has the following format:

Offset	Name	Meaning
0	NextBlock	Pointer to next connected Lock or Null.
4	DiskBlock	Block-Nr. of the directory or file header.
8	AccessType	Access type: -1= excl. access, -2= general access.
12	ProcessID	Identification number.
16	VolNode	Pointer to disk information.

This structure represents the key to this file or the sub-directory because it can be accessed with it (see the Makedir command of the CLI).

Lock

```
lock = Lock ( Name, Mode )
D0     -84     D1    D2
```

Finds a file key

Find a file or a sub-directory with the name Name on the disk and create a structure. The Mode determines what type of access can occur on this file. If it is reading (-2), several tasks can be read from this file. If it is writing, (-1), only this program can write into the file.

CurrentDir

```
oldLock = CurrentDir ( Lock )
  D0       -126          D1
```

Elevate sub directory to current directory

The sub-directory specified by Lock is elevated to the current directory (see the CD command of the CLI).

The value returned represents the pointer to the previous directory, the Lock.

ParentDir

```
Lock_neu = ParentDir ( Lock )
  D0        -210         D1
```

Determines the highest level directory

The directory indicated by Lock is determined and its Lock is returned in D0. If Lock already belongs to the highest directory (Root directory), a null is returned in D0.

DeleteFile

```
Status = DeleteFile ( Name )
  D0      -72           D1
```

Deletes a file

The file with the indicated name is deleted. The name must be text which is terminated with a null byte. An error message is returned in D0 if the function could not be performed (for example, file not present, file write protected, directory not empty).

If a sub-directory is indicated for deletion, no entries can still be present in the sub-directory.

Rename

```
Status = Rename ( Name_old, Name_new )
  D0      -78        D1          D2
```

Renames a file

The file or directory with the name provided in "Name_old" is renamed. If a file with that name already exists, the operation is interrupted and an error indication is returned.

The two name indications can also contain paths. In this case the file is brought from the old directory into the new directory with the new name. This can only be done on the same disk.

DupLock

```
newLock = DupLock ( Lock )
D0        -96        D1
```

Copies a lock

The old Lock structure is copied into a new structure. D0 then points to the new structure. This can be used if several processes should access this file. No Lock can be copied if it's only authorized for writing since it is already authorized for an exclusive access.

UnLock

```
UnLock ( Lock )
-90       D1
```

Removes a lock

The Lock structure which was created with Lock, DupLock or CreateDir, is removed and the memory occupied is released again.

Examine

```
Status = Examine ( Lock, InfoBlock )
  D0      -102       D1     D2
```

Gets file information

The structure to which D2 points is filled with information about the file specified. This structure is called FileInfoBlock and appears as follows:

Offset	Name	Description
0	DiskKey.L	Disk number.
4	DirEntryType.L	Entry type (+=Directory, -=file).
8	FileName 108	Bytes with the filename.
116	Protection.L	File protected?
120	EntryType.L	Entry type.
124	Size.L	File length in bytes.
128	NumBlocks.L	Number of blocks occupied.
132	Days.L	Creation date.
136	Minute.L	Creation time.
140	Tick.L	Creation time.
144	Comment 116	Bytes with comments.

D0 contains a 0 if the function could not be performed.

ExNext

```
Status = ExNext ( Lock, InfoBlock )
  D0      -108      D1     D2
```

Determines the next directory entry

The InfoBlock filled with Examine and the Lock of the selected directories is passed to this function. The information of the first suitable entry from this directory is entered into the InfoBlock. During another call of ExNext, a search is made for the next entry of this directory and its information is returned. If a further entry cannot be found, or an error has occurred, a null is returned in D0. The table of contents of a disk can be read with the Lock, Examine and ExNext commands.

The path is as follows:

1.) The key to the desired directory is created with Lock.

2.) The directory name or the name of the disk can be determined with Examine. At the same time the FileInfoBlock is created which is necessary for the next function.

3.) The individual entries in the directory are read with repeated calls of the ExNext function. This information is entered into the FileInfoBlock. This is repeated until the ExNext function returns a null. At that point no additional entries are available!

Following is a small machine language subroutine which completes these steps. The Print routine that is called is not presented here. It could for example print the name and length of file just read to the screen.

Before the call of this routine the DOS library must be opened and the DOS base address must be stored in "dosbase."

```
Lock       = -84
Examine    = -102
ExNext     = -108

IoErr      = -132

     ...
directory:                            ;* Table of Content of DF0:
     move.l  dosbase,a6               ;DOS-Base address in A6
     move.l  #name,d1                 ;pointer to Path-/Filename
     move.l  #-2,d2                   ;Mode "Read"
     jsr     Lock(a6)                 ;search for file
     tst.l   d0                       ;found ?
     beq     Error                    ;no !
     move.l  d0,locksav               ;otherwise save key

     move.l  dosbase,a6               ;DOS-Base address
     move.l  locksav,d1               ;Key in D1
     move.l  #fileinfo,d2             ;pointer to FileInfoBlock
```

```
     jsr      Examine(a6)        ;Get Disk-Name
     tst.l    d0                 ;OK?
     beq      error              ;no (occurs rarely)
     bra      output             ;otherwise output Name

loop:                            ;* Read Filenames
     move.l   dosbase,a6         ;DOS-Base address
     move.l   locksav,d1         ;Key in D1
     move.l   #fileinfo,d2       ;pointer to FileInfoBlock
     jsr      ExNext(a6)         ;search for next file
     tst.l    d0                 ;found ?
     beq      error              ;no: End

output:                          ;* Output Name
     bsr      Print              ;Output/evaluate Name etc.
     bra      loop               ;and continue ...

error:                           ;* Determine I/O-Status
     move.l   dosbase,a6         ;DOS-Base address in A6
     jsr      IoErr(a6)          ;Get Status
     rts                         ;End...

name:     dc.b  'DF0:',0
 align                           ; some assembler use even
locksav:  blk.l   0
fileinfo: blk.l 260
 end
```

After termination of this routine an error code which was determined by the IoErr function is returned in D0. This code should be 232 (no_more_entries) or something went wrong.

Info

```
Status = Info ( Lock, InfoData )
  D0      -104    D1     D2
```

Gets disk information

The parameter block, to which D2 points, is filled with information about the disk in use. This block must start at an address which is divisible by four (longword aligned).

Lock must fit the disk, a file or a subdirectory of this disk.

The parameter block InfoData has the following format:

Offset	Name	Description
0	NumSoftErrors	Number of disk errors.
4	UnitNumber	Installed disk drive.
8	DiskState	Disk status (see below).
12	NumBlocks	Number of blocks on the disk.
16	NumBlocksUsed	Number of blocks used.
20	BytesPerBlock	Number of bytes per block.
24	DiskType	Disk type (see below).
28	VolumeNode	Pointer to disk name.
32	InUse	<>0, if disk is active.

DiskState shows the status of the disk. The possible results are:

80	Disk is write protected.
81	Disk is under repair (validating).
82	Disk OK and can be written.

DiskType contains the disk type as text is inserted. The possible values are:

-1	No disk inserted.
BAD	Disk not readable (wrong Format).
DOS	DOS disk.
NDOS	Format OK, but not a DOS disk.
KICK	Kickstart disk.

SetComment

```
Status = SetComment ( Name, Comment )
  D0       180          D1     D2
```

Sets a file comment

The file or the sub-directory Name is given a comment. The comment can be up to 80 characters in length and must terminate with a null byte.

SetProtection

```
Status = SetProtection ( Name, Maske )
  D0      -186             D1     D2
```

Sets the file status

The write or read Status of the file indicated, or of the sub-directory is set. The lower 4 bits of the mask have the following significance:

Bit	Significance when set
0	file not erasable
1	not executable
2	not to be overwritten
3	not readable

5.4.3 Process processing

CreateProc

```
Process = CreateProc  (Name, Pri, Segment, Stack)
  D0       -138         D1    D2     D3       D4
```

Creates a new process

A new Process structure is created under the name to which D1 points. This process runs under the priority indicated in Pri and gets a Stack of the size specified in Stack.

A pointer to the Segment list is passed in Segment (see also LoadSeg), in which the program code to be started is defined. The program should start in the first segment of the list.

The result of the function is the new Process ID or a 0, if an error occurred.

DateStamp

```
DateStamp ( Vector )
 -192         D1
```

Determines the date and time

In D1 a pointer is returned to a table of three long words. If the time was not set in the Amiga all of these long words contain a 0. Otherwise the first long word contains the number of days passed since January 1978, the second the number of minutes passed since midnight, and the third the 1/50 seconds elapsed in this minute. This value is always a multiple of 50 so that the number of seconds *50 is always indicated.

Delay

```
Delay ( Time )
-198     D1
```

Stops the execution of the current process for a short period of time

The executing process is stopped for the number of 1/50 seconds indicated in Time.

DeviceProc

```
Process = DeviceProc ( Name )
  D0       -174          D1
```

Identifies the Process using I/O

The identification of the process which at this moment uses the Input/Output channel indicated in Name is returned, or a 0 if a process wasn't found.

If the name relates to a channel which is on a disk, a pointer to the Lock structure of the corresponding directory can be maintained with the IoErr function.

Exit

```
Exit ( Parameter )
-144    D1
```

Terminates a program

The executing program is terminated. If the program was called from the `CLI`, control is returned to it and the integer value in Parameter is interpreted as a return value. If the program was started as Process, this process is erased through Exit and the Stack, Segment and Process memory used by it is released again.

Execute

```
Status = Execute (Command, Input, Output)
  D0      -222        D1      D2     D3
```

Calls a CLI command

The CLI commands which are provided in a text file and to which D1 points, is executed. With Input and Output the I/O of the CLI command can be redirected but their handle must be indicated here. If a null is indicated for Input or Output, the standard channel is used.

LoadSeg

```
Segment = LoadSeg ( Name )
  D0       -150      D1
```

Loads a program file

The program file Name is loaded into memory. The program can be spread over several memory modules if not enough memory space is available. The segments thus created are chained together by having the first entry of every segment a pointer to the next segment of the list. If this pointer is 0, this is the last segment.

If an error occurs during this process, all previously loaded segments are released again and a 0 is returned in D0. Otherwise D0 contains a pointer to the first segment.

The loaded program can only be started with CreateProc or erased with UnLoadSeg.

UnLoadSeg

```
UnLoadSeg ( Segment )
-156         D1
```

Erases a program file which was loaded

The program file which was loaded with LoadSeg is erased and the memory used is released again. The pointer in D1 points to the first segment of the list (see LoadSeg).

GetPacket

```
Status = GetPacket ( Waitflag )
  D0      -162          D1
```

Gets a packet

Gets a packet which was sent by another process. If the Waitflag is true (-1), a wait occurs for the content of the packet, otherwise no wait occurs and a null is returned if a packet isn't available.

QueuePacket

```
Status = QueuePacket ( Packet )
 D0      -168             D1
```

Sends a packet

The packet, to which D1 points, is sent. If no error occurs, the value <>0 is returned in D0.

5.5 DOS error messages

DOS error messages

The following list contains the error codes and their meaning from IoErr or the Why command of the `CLI`.

103 Insufficient free store

Not enough storage is available.

104 Task table full

Already 20 processes are active. No more are permitted.

120 Argument line invalid or too long

The argument list for this command is not correct or contains too many arguments.

121 File is not an object module

The file called is not capable of being executed.

122 Invalid resident library during load

The resident library called is invalid.

202 Object in use

The indicated file or the directory is being used at this moment by another program and is not available for other applications.

203 Object already exists

The filename indicated already exists.

204 Directory not found

The selected directory does not exist.

205 Object not found

The channel with the name indicated does not exist.

206 Invalid window

The parameters for the window to be opened are not correct.

209 Packet requested type unknown

The desired function is not possible on the device indicated.

210 Invalid stream component name

The filename is invalid (too long or has unauthorized characters).

211 Invalid object lock

The Lock structure indicated is invalid.

212 Object not of required type

File and directory name have been reversed.

213 Disk not validated

The disk is either not recognized yet by the system or is defective.

214 Disk write-protected

The disk is write protected.

215 Rename across devices attempted

The Rename function is possible only within a disk.

216 Directory not empty

A directory which is not empty cannot be erased.

218 Device not mounted

The selected disk is not mounted.

219 Seek error

Seek function with illegal parameters.

220 Comment too big

The comment for the file is too large.

221 Disk full

The disk is full or doesn't contain enough free space for the application.

222 File is protected from deletion

The file is protected against deletion.

223 File is protected from writing

The file is protected against writing.

224 File is protected from reading

The file is protected against reading. With the last three error messages the List command can be used to check the status of the affected file.

225 Not a DOS disk

This disk was not formatted with AmigaDOS format.

226 No disk in drive

The drive does not contain a disk.

232 No more entries in directory

The last ExNext function could not detect a suitable entry in the directory.

6.
File Control

6. File Control

The Filesystem

File control in the Amiga is performed by the *Filesystem*. The Filesystem is a separate task which is addressed by DOS when files must be handled (for example programs).

The Filesystem differentiates to which device file access should be directed (for example disk or hard disk) and addresses the device drivers to access the selected mass storage. The selection of the devices is not important for the use of Filesystem. It works with every device capable of using data blocks.

With this method it's possible to interface many different mass storage devices provided a suitable device driver is included which can work with the Filesystem. By using the Filesystem, the system is not tied to a certain fixed device for file control and can be enhanced with additional devices with little effort.

Path of data access through Filesystem

If access of a file is needed from a device, the command is normally sent first to DOS (for example read program XY from disk). This determines that it is a file access and sends the command to the Filesystem. The Filesystem then controls access by determining how many blocks from which device should be addressed. The commands for writing and reading of blocks are passed by the Filesystem to the proper device drivers which then communicate directly with the hardware.

An access is a very complicated procedure which unfortunately also consumes much time. This disadvantage was accepted by the systems developers to keep the system flexible.

To reduce the speed loss, the Filesystem stores the last blocks read in RAM so that during new accesses they can be read from the faster RAM. The number of blocks which can be stored in RAM, can be enlarged with the `CLI` command Addbuffers.

6.1 The disk monitor

Before examining the various block types which can be found on the disk, the use of a disk monitor is discussed.

The description of the monitor which follows cover the one presented in the Appendix of this book. The source file is found in the Appendix.

The monitor is loaded and started from the optional disk for this book using the `CLI` by entering DiskMon. It accesses the internal drive (DF0). If another drive is accessed, it can indicated at the start as a parameter in the command line. To access DF1, "DiskMon df1:" must be input for the start.

Due to differences in screen sizes between PAL (Europe) and NTSC (US) Amigas, the disk monitor program for the NTSC machines displays either the ASCII data or HEX output in the same area. PAL systems can display more lines so both ASCII and HEX output may be displayed on the screen. The source code contains comments which describe the changes required for each system. The optional disk contains the NTSC version.

6.1.1 The commands of the monitor

The characters enclosed in brackets indicate the keys to be activated ([#] for input of the character: #).

[Esc] Used to leave the monitor.

[#] The block number to be read can be indicated. The input must be in decimal and must always have four digits (for example 0013 to read block 13). If an incorrect number is indicated, the input must be repeated.

[$] Has the same meaning as [#] with the difference that the block number must be input in hexadecimal.

[+] The next logical block is loaded and displayed.

[-] The previous logical block is loaded and displayed.

[R] The current block is read again (for instance if the disk is changed).

[W] The block in the buffer is written to disk.

[C] The checksum of the block is calculated and stored. The sum is also displayed. The command is not suitable for the calculation of the boot block checksum.

[A] The cursor jumps into the data display of the block in ASCII and permits the editing of the blocks in ASCII. After the word buffer, the current cursor position in the block is displayed. A quick glance can determine if even or odd addresses are being edited. This sub-point is left for the main menu with [Esc].

[H] The display changes to hexadecimal and permits editing of the block in bytes. Two characters (1 byte) must always be input. Otherwise editing is similar to ASCII.

6.2 The various block types

Aside from the blocks containing program data, there are other blocks on the disk which mark individual files and connect them with each other for better data control. There is also an item called the boot block.

6.2.1 The boot block

The boot block is used by the operating system to indicate if the disk can be started during initialization like the Workbench disk. The name boot block is misleading, since there are actually two blocks. These are the lowest two blocks of the disk (block 0 and 1, consisting of 1024K). To allow the disk to start, certain data must be written into it by using the Install command.

The boot block can also be used to call machine language programs which are executed as soon as the disk is inserted into the internal drive (DF0) after a reset. To use this capability fully, it's necessary to understand how the operating system calls the boot block. This is discussed in more detail after we look at the construction of the boot blocks.

Construction of the boot block

Longword 1 contains:

The ASCII identification of the disk terminated with null
(valid only for DOS).

The ASCII identification can be either for DOS (for a DOS disk)
or KICK (for a Kickstart disk).

If the DOS identification cannot be found, the message "No DOS Disk" is output.

Longword 2 contains:

The checksum of the boot block.

Longword 3 contains:

A pointer to the root block (normally $370=880).
The pointer does not have to be set.

Example for the construction of the header for the boot block:

```
Identification: dc.b "DOS",0 ;ASCII identification of the disk
Chksum:  dc.l $????????      ;Checksum for boot block
RootBlk: dc.l $00000370      ;pointer to Root block
```

Starting at the fourth longword (the 12th byte) is the actual boot program which is executed when the checksum is correct. Normally the following routine is stored here. This can be replaced by the user, unless the original one is also executed. This is shown in the documentation of the boot routine which follows.

The program which is written by the Install command appears as follows:

```
BootPrg: lea     Resname(PC),A1 ;pointer to name of the
                                 Resident structure
         jsr     -96(A6)        ;search Resident structure
         tst.l   D0             ;test for error
         beq     Error          ;if an error occurred
         move.l  D0,A0          ;pointer to Resident structure
                                ;after A0
         move.l  22(A0),A0      ;pointer to the Initialization
                                 after A0
         moveq   #$00,D0        ;clear D0 if no error occurred
Ende:    rts                    ;Return jump to the Boot
                                ;Routine of ROM
```

Following this is a routine which is called if an error occurs (which normally should not happen):

```
Error:   moveq   #$ff,D0    ;load D0 with $ff (for error)
         bra     Ende       ;terminate program
```

The following bytes contain the name of the desired Resident structure (in ASCII):

```
Resname: dc.b "dos.library",0
```

This routine searches for the Resident structure to construct the DOS library and passes its base address in A0 to the boot routine in the operating system.

Calculating the boot block checksum

Let's examine the routine which tests the boot block checksum.

On entry to the routine, the pointer of the boot block which is already in RAM, is in A0. First a 256 is written into D1. This is used as a counter for the number of longwords for which the checksum should be formed. Since a longword has 4 bytes, there are 4x256 = 1024 bytes, which is exactly the length of two disk sectors (in this case 0 and 1).

```
fe8a14 move.w  #$00ff,D1
```

Then the register which is later used for the addition, is cleared.

```
fe8a18 moveq   #$00,D0
```

Now a longword from the disk buffer (which contains the boot block) is added to D0.

```
fe8a1a add.l    (A0)+,D0
```

A test is made for an overflow. If one didn't occurr, a jump occurs to $fe8a20.

```
fe8a1c bcc.s    $fe8a20
```

If an overflow occurred, it is added to D0.

```
fe8a1e addq.l   #1,D0
```

A test is made to see if the whole disk buffer was added and if not, addition continues.

```
fe8a20 dbf      D1,$fe8a1a
```

The following command reverses all bits, which means the bits which were erased are set and vice versa.

```
fe8a24 not.l    D0
```

Finally a test is made if all bits have been erased. If this is not the case, the checksum is false and a branch is taken to $fe8a5c to wait for the insertion of a bootable disk.

```
fe8a26 bne.s    $fe8a5c
```

The boot checksum is a longword addition with overflow.

6.2.2 The calculation of the user's boot checksum

The calculation of the checksum is fairly simple. First the data which are stored in the boot block must be prepared as follows:

Bootbuffer is the label which indicates the start of the user's data buffer.

```
Bootbuffer:
             dc.b "DOS",0        ; DOS-identification
             dc.l 0              ; Checksum
             dc.l $00000370      ; pointer to Rootblock

Program:   From here on the executable program is stored.
```

With the following program the sum of the user's data can be formed, starting at the DOS identification. The result of this is then reversed by the program and entered as checksum (Offset 4 starting from the boot buffer). It is important that the sum entered before the calculation is null. This is done by the program.

During the calculation of the user's data it can be determined that the checksum is correct ($00000000) and is recognized as such by the operating system.

Program for the calculation of the checksum:

```
                lea Bootbuffer,A0
                lea 4(a0),a1
                clr.l (a1)
                move.w  #$00ff,D1
                moveq   #$00,D0
Loop:           add.l   (A0)+,D0
                bcc     Jump
                addq.l  #1,D0
Jump:           dbf     D1,Loop
                not.l   D0
                move.l  D0,(a1)
```

All that remains is deciding what the boot program contains and writing it to the disk.

The Boot routine

How the operating system implements the booting of disks is explained in the following section. To explain the entire reset routine would be too extensive, so the explanation is limited to the most important parts.

1.) Creating the Resident structure table $fc0504

```
fc0504 bsr.l  $fc0900
```

This call causes a search for all Resident structures (reset proof programs) which are in ROM. The pointers to these structures are stored in a table. The sequence of the entries in the table is according to priority of the Resident structures. The pointer to the table is stored at Offset 300 of the Execbase structure (ResModules).

2.) Processing the Resident structures $fc0522

```
fc0522 bsr.l  $fc0af0
```

After the creation of the table, the InitCode() routine is called. It processes the Resident structures in the sequence of their priorities (high priority has precedence over lower priority). The lowest priority is -60 and the structure which pertains to it is the routine for the booting of the disk.

Now for the documented Boot routine of ROM:

```
fe88d6 movem.l A5-A3/D3-D2,-(A7) Save Register
fe88da moveq   #$00,D3           Clear D3
fe88dc suba.l  A4,A4             Clear A4
```

```
fe88de lea      $fe8b3a,A3          Pass the pointer
                                    to RTS in A3
fe88e4 link     A5,#-126            increase stack
fe88e8 suba.l   #$0000007e,A5       A5 to beginning of stack
fe88ee move.l   A6,0(A5)            enter Execbase
fe88f2 move.l   D3,4(A5)            enter null
```

The following routine reserves the storage for the disk buffer.

```
fe88f6 move.l   #$00000488,D0  pass the size of the storage to use
fe88fc move.l   #$00010002,D1  order Chip-Memory and erase

fe8902 jsr      -198(A6)       Alloc Mem
fe8906 tst.l    D0             did error occur?
fe8908 bne.s    $fe8924        if not, branch
fe890a movem.l A6-A5/D7,-(A7) save Register
fe890e move.l   #$30010000,D7  pass error number
fe8914 move.l   $0004,A6       write Execbase in A6

fe8918 jsr      -108(A6)       Alert
fe891c movem.l (A7)+,A6-A5/D7 restore register
fe8920 bra.l    $fe8b2a        reduce stack
                               and end

fe8924 move.l   D0,A4          pass the pointer
                               to reserved
                               storage in A4
fe8926 lea      $fe889e,A0     pass pointer to
                               ASCII "STrap."
fe892c move.l   A0,54(A5)      as Disk-I/O-Name
                               and store

fe8930 move.l   A0,102(A5)     enter as Portname in
                               Message-Port-
                               List
fe8934 suba.l   A1,A1          erase A1
fe8936 jsr      -294(A6)       get running task
fe893a move.l   D0,108(A5)     and store pointer
fe893e move.b   #$00,106(A5)   enter flags in the Msg.- Port-
                               Structure
A blank List is created.
fe8944 lea      112(A5),A0     pass pointer to
                               Message-List to
                               A0
fe8948 move.l   A0,(A0)        write pointer to
                               Message-List in
                               ml_Head
fe894a addq.l   #4,(A0)        set ml_Head to ml_Tail

fe894c clr.l    4(A0)          set ml_Tail to 0
fe8950 move.l   A0,8(A0)       set ml_TailPred to
                               ml_Head
fe8954 moveq    #$ff,D0        search for signal of the running
                               Task
fe8956 jsr      -330(A6)       Alloc Signal
fe895a move.b   D0,107(A5)     enter Signal into Msg.- Port-
                               Structure
fe895e bpl.s    $fe897a        if everything O.K.
                               branch
fe8960 movem.l A6-A5/D7,-(A7) otherwise no Signal
                               available
fe8964 move.l   #$30070000,D7  pass error number
```

```
fe896a move.l  $0004,A6        write Execbase in A6

fe896e jsr     -108(A6)        Alert
fe8972 movem.l (A7)+,A6-A5/D7  restore Register
fe8976 bra.l   $fe8b1e         branch
fe897a lea     92(A5),A0       get pointer to Msg.-Port-
                               Node
fe897e move.l  A0,58(A5)       and enter into Disk-I/O-Structure
fe8982 lea -192(PC)(=$fe88c4),A0  pass pointer to ASCII
                                  "trackdisk.device"

fe8986 lea     44(A5),A1        pass pointer to I/O-Request-
                                Structure
fe898a moveq   #$00,D0          set drive DF0

fe898c moveq   #$00,D1          do not pass Flags
fe898e jsr     -444(A6)         open Trackdisk-Device

fe8992 tst.l   D0               Device open ?
fe8994 beq.s   $fe89b0          if yes, branch
fe8996 movem.l A6-A5/D7,-(A7)   otherwise save Register
                                and
fe899a move.l  #$30048014,D7    pass Error number
fe89a0 move.l  $0004,A6         write Execbase in A6

fe89a4 jsr     -108(A6)         Alert
fe89a8 movem.l (A7)+,A6-A5/D7   restore Register
fe89ac bra.l   $fe8b14          Free Signal and End
```

The following routine clears all buffers.

```
fe89b0 move.w  #$0100,$dff096        block DMA accesses
fe89b8 lea     44(A5),A1             pass the pointer
                                     to the I/O-Request-
                                     Structure in A1
fe89bc move.w  #$0005,28(A1)         pass command:
                                     Clear Request
fe89c2 jsr     -456(A6)              DO IO
fe89c6 tst.l   D0                    did error occur?
fe89c8 bne.l   $fe8ac8               if yes, branch
```

In the next routine the number of the disk changes is passed in D2.

```
fe89cc lea     44(A5),A1        pass pointer to
                                I/O-Request-Structure
fe89d0 move.w  #$000d,28(A1)    pass command:
                                Change Number
fe89d6 jsr     -456(A6)         DO IO
fe89da tst.l   D0               did error occur?
fe89dc bne.l   $fe8ac8          if yes, branch
fe89e0 move.l  76(A5),D2        otherwise pass the number
                                of disk changes in D2
```

Now a test is made if the disk inserted is a DOS disk.

```
fe89e4 lea     44(A5),A1             pointer to
                                     Structure
fe89e8 move.w  #$0002,28(A1)         command passed:
                                     Read
fe89ee move.l  #$00000400,36(A1)     pass length
fe89f6 move.l  A4,40(A1)             pass pointer to
```

```
                                           Data Buffer
fe89fa move.l  #$00000000,44(A1)           pass Offset for
                                           Boot-Block
fe8a02 jsr     -456(A6)                    DO IO
fe8a06 tst.l   D0                          error occurred ?
fe8a08 bne.s   $fe8a5c                     if yes, branch
fe8a0a move.l  (A4),D0                     first data to D0
fe8a0c cmp.l   -334(PC)(=$fe88c0),D0       compare with DOS-
                                           identification
fe8a10 bne.s   $fe8a5c                     if no agreement
                                           branch
fe8a12 move.l  A4,A0                       otherwise pass the pointer
                                           the data buffer in A0
```

Next follows the routine for testing of the boot block checksum.

```
fe8a14 move.w  #$00ff,D1    load counter with value for
                            1024K(Boot Block)
fe8a18 moveq   #$00,D0      clear Register for
                            Checksum
fe8a1a add.l   (A0)+,D0     add content of longword from
                            Diskbuffer to D0
fe8a1c bcc.s   $fe8a20      if no overflow was created
                            branch
fe8a1e addq.l  #1,D0        otherwise increment D0 by
                            overflow
fe8a20 dbf     D1,$fe8a1a   if the 1024K
                            of the Boot-Block have not been
                            processed, add further
fe8a24 not.l   D0           reverse the Bits of the Checksum
fe8a26 bne.s   $fe8a5c      if Checksum is not
                            O.K., branch
```

Jump to the Boot program.

```
fe8a28 lea      44(A5),A1          pass pointer to I/O-Request-
                                   Structure
fe8a2c jsr      12(A4)             jump into the Boot-Program

fe8a30 tst.l    D0                 error occurred ?
fe8a32 beq.s    $fe8a56            if not, branch
fe8a34 move.l   D0,-(A7)           write error register to the
                                   Stack
fe8a36 move.l   A7,A1              write Alert-Parameter to
                                   A1
fe8a38 movem.l  A6-A5/D7,-(A7)     save Register
fe8a3c move.l   #$30000001,D7      pass Error number
fe8a42 lea      (A1),A5            pass Alert-Parameter to A5
fe8a44 move.l   $0004,A6           write Execbase in A6

fe8a48 jsr      -108(A6)           Alert
fe8a4c movem.l  (A7)+,A6-A5/D7     restore Register
fe8a50 addq.l   #4,A7              set Stack beginning after
                                   Alert-Parameter
fe8a52 bra.l    $fe8b00            branch
```

Next the pointer to the initialization routine is passed to A3.

```
fe8a56 move.l  A0,A3   Pass pointer to the initialization routine
fe8a58 bra.l   $fe8b00 branch
```

This routine tests if the display asking for insertion of the Workbench disk has been sent to the screen already. If not, it is displayed and then the motor is switched off.

```
fe8a5c move.l  4(A5),D0          Gfx-Library available ?
fe8a60 bne.s   $fe8a66           if yes, branch
fe8a62 bsr.l   $fe8b7e           else Gfx-Library
                                 open
fe8a66 move.w  #$8100,$dff096    permit DMA accesses
fe8a6e lea     44(A5),A1         pass pointer to I/O-Request-
                                 Structure in A1
fe8a72 move.w  #$0009,28(A1)     pass command:
                                 Motor
fe8a78 clr.l   36(A1)            switch off motor
fe8a7c jsr     -456(A6)          DO IO
fe8a80 tst.l   D0                Error occurred ?
fe8a82 bne.s   $fe8ac8           if yes, branch
```

The following routine waits until the disk is changed.

```
fe8a84 lea     44(A5),A1      pass pointer to I/O-Request-
                              Structure to A1

fe8a88 move.w  #$000d,28(A1)  pass command:
                              Change Num
fe8a8e jsr     -456(A6)       DO IO
fe8a92 tst.l   D0             Error occurred ?
fe8a94 bne.s   $fe8ac8        if yes, branch
fe8a96 cmp.l   76(A5),D2      Disk changed in the meantime ?

fe8a9a beq.s   $fe8a84        if not, continue testing
```

The program now senses if a disk is inserted.

```
fe8a9c lea     44(A5),A1      pass pointer to I/O-Request-
                              Structure
fe8aa0 move.w  #$000e,28(A1)  pass command:
                              Changestate
fe8aa6 jsr     -456(A6)       DO IO
fe8aaa tst.l   D0             Error occurred?
fe8aac bne.s   $fe8ac8        if yes, branch
fe8aae tst.l   76(A5)         Disk inserted ?
fe8ab2 bne.s   $fe8a9c        if not, continue testing
fe8ab4 bra.l   $fe89b0        if Disk was inserted, branch
```

The number of disk changes is passed.

```
fe8ab8 lea     44(A5),A1      pass pointer to I/O-Request-
                              Structure
fe8abc move.w  #$000d,28(A1)  pass command:
                              Change Num
fe8ac2 jsr     -456(A6)       DO IO
fe8ac6 bra.s   $fe8a5c        branch
```

A jump to the following routine occurs in case of error.

```
fe8ac8 cmpi.b  #$1d,75(A5)    Error occurred
fe8ace beq.s   $fe8ab8        if yes, branch
fe8ad0 pea     $0000          write $0000 into
```

```
                                 the stack
fe8ad4 move.w   72(A5),2(A7)     get I/O command and write into stack
fe8ada pea      $0000            write $0000 into the
                                 stack
fe8ade move.b   75(A5),3(A7)     get I/O error and write into stack
fe8ae4 move.l   A7,A1             pass stack as Alert-
                                  Parameter
fe8ae6 movem.l  A6-A5/D7,-(A7)    save Register
fe8aea move.l   #$30068014,D7     pass error number
fe8af0 lea      (A1),A5           pass Alert-Parameter
                                  A5
fe8af2 move.l   $0004,A6          write Execbase in A6

fe8af6 jsr      -108(A6)          Alert
fe8afa movem.l  (A7)+,A6-A5/D7    restore Register

fe8afe addq.l   #8,A7             set stack beginning behind
                                  Alert-Parameter
```

During the following routine everything is restored to normal and a jump is performed to the initialization routine of DOS.

```
fe8b00 bsr.l    $fe8dd0             branch
fe8b04 move.w   #$8100,$dff096      permit DMA accesses
fe8b0c lea      44(A5),A1           pass pointer to I/O-Request-
                                    Structure
fe8b10 jsr      -450(A6)            Close Device
fe8b14 moveq    #$00,D0             erase D0
fe8b16 move.b   15(A5),D0           pass Signal number
fe8b1a jsr      -336(A6)            Free Signal
fe8b1e move.l   A4,A1               write beginning address of
the                                  occupied storage to A1
fe8b20 move.l   #$00000488,D0       pass size of occupied storage
fe8b26 jsr      -210(A6)            Free Mem
fe8b2a adda.l   #$0000007e,A5       set A5 to end of stack
fe8b30 unlk     A5                  bring stack to normal size
fe8b32 move.l   A3,A0               pass pointer to initialization
                                     routine in A0
fe8b34 movem.l  (A7)+,A5-A3/D3-D2   restore Register
fe8b38 jmp      (A0)                perform initialization
routine
fe8b3a rts
```

In the normal case, A0 contains the pointer to the initialization routine. In case of an error, A0 contains the pointer to $fe8b3a. A jump is made through WarmCapture (if it contains a value).

This routine opens the Gfx library.

```
fe8b7e lea      -68(PC)(=$fe8b3c),A1  pass pointer to ASCII
                                      "graphics.library"
                                      to A0
fe8b82 moveq    #$00,D0               pass Version
fe8b84 jsr      -552(A6)              Open Library
fe8b88 move.l   D0,4(A5)              enter Gfx-Base in the stack

fe8b8c bne.s    $fe8ba6               if no error occurred,
branch
fe8b8e movem.l  A6-A5/D7,-(A7)        save Register
fe8b92 move.l   #$30038002,D7         pass error number
```

```
fe8b98 move.l  $0004,A6                write Execbase in A6
fe8b9c jsr     -108(A6)                Alert
fe8ba0 movem.l (A7)+,A6-A5/D7          restore Register
fe8ba4 rts
```

The following routine reserves storage for the View port.

```
fe8ba6 movem.l A6/A3-A2/D5-D2,-(A7) save Register
fe8baa move.l  0(A5),A6             pass Execbase to A6
fe8bae move.l  #$00005e9a,D0        write Byte length in D0
fe8bb4 move.l  #$00010003,D1        request chip memory, erase
                                   and not relocatable
fe8bba jsr     -198(A6)             Allocate memory
fe8bbe tst.l   D0                   error?
fe8bc0 bne.l   $fe8be0              if not, branch
fe8bc4 movem.l A6-A5/D7,-(A7)       save Register
fe8bc8 move.l  #$30010000,D7        pass error number
fe8bce move.l  $0004,A6             write Execbase to A6
fe8bd2 jsr     -108(A6)             Alert
fe8bd6 movem.l (A7)+,A6-A5/D7       restore Register
fe8bda movem.l (A7)+,A6/A3-A2/D5-D2 restore Register
fe8bde rts
```

In the routines which follow, the structures required for the display output are created, among other things.

```
fe8be0 move.l  D0,8(A5)           store pointer to View-Port
fe8be4 addi.l  #$00000028,D0      reserve space
fe8bea move.l  D0,12(A5)          store pointer to View
fe8bee addi.l  #$00000012,D0      reserve space
fe8bf4 move.l  D0,16(A5)          store pointer to Rast-Port
fe8bf8 addi.l  #$00000064,D0      reserve space
fe8bfe move.l  D0,20(A5)          store pointer to TmpRas
fe8c02 addi.l  #$00000008,D0      reserve space
fe8c08 move.l  D0,24(A5)          store pointer to RasInfo
fe8c0c addi.l  #$0000000c,D0      reserve space
fe8c12 move.l  D0,28(A5)          store pointer to Bitmap
fe8c16 addi.l  #$00000028,D0      reserve space
fe8c1c move.l  D0,32(A5)          store PlanePTR 1-4
fe8c20 move.l  #$00001f40,D1      set Byte length
fe8c26 add.l   D1,D0              reserve space
fe8c28 move.l  D0,36(A5)          store PlanePTR 5-8
fe8c2c add.l   D1,D0              reserve space
fe8c2e move.l  D0,40(A5)          store pointer to Buffer
fe8c32 move.l  4(A5),A6           write Gfx-Base in A6
fe8c36 move.l  8(A5),A0           pass pointer to View-Port in A0
fe8c3a jsr     -204(A6)           Init View Port
fe8c3e move.l  12(A5),A1          pass pointer to View in A1
fe8c42 jsr     -360(A6)           Init View
fe8c46 move.l  28(A5),A0          write pointer to Bitmap-
                                  Structure in A0
fe8c4a moveq   #$02,D0            pass 2 Bitplanes
fe8c4c move.l  #$00000140,D1      pass width
fe8c52 move.l  #$000000c8,D2      pass heigth
fe8c58 jsr     -390(A6)           Init Bitmap
fe8c5c move.l  28(A5),A0          write pointer to Bitmap-
                                  Structure in A0
fe8c60 move.l  32(A5),8(A0)       store PlanePTR 1-4
fe8c66 move.l  36(A5),12(A0)      store PlanePTR 5-8
fe8c6c move.l  16(A5),A1          pass pointer to Rast-Port in A1
fe8c70 jsr     -198(A6)           Init Rast-Port
```

```
fe8c74 move.l  20(A5),A0            write pointer to TmpRas in A0
fe8c78 move.l  40(A5),A1            write pointer to Buffer in A1
fe8c7c move.l  #$00001f40,D0        pass size in D0
fe8c82 jsr     -468(A6)             Init Tmp Ras
fe8c86 move.l  24(A5),A0            pass pointer to RasInfo-
                                    Structure in A0
fe8c8a move.l  28(A5),4(A0)         store pointer to Bitmap
fe8c90 move.l  16(A5),A0            pass pointer to Rast-Port-
                                    Structure in A0
fe8c94 move.l  28(A5),4(A0)         store pointer to Bitmap
fe8c9a move.l  20(A5),12(A0)        store pointer to Tmp Ras
fe8ca0 move.l  8(A5),A0             write pointer to View-Port-
                                    Structure to A0
fe8ca4 move.w  #$00c8,26(A0)        store DHeigth
fe8caa move.w  #$0140,24(A0)        store DWith
fe8cb0 move.l  24(A5),36(A0)        store pointer to RasInfo
fe8cb6 clr.w   32(A0)               erase Modes
fe8cba move.l  12(A5),A3            write pointer to View-
                                    Structure to A3
fe8cbe move.l  8(A5),0(A3)          store pointer to View-Port
fe8cc4 move.l  A3,A0                pass pointer to View in A0
fe8cc6 move.l  8(A5),A1             pass pointer to View-Port
                                    in A1
fe8cca jsr     -216(A6)             Make View-Port
fe8cce move.l  A3,A1                write pointer to View to A1
fe8cd0 jsr     -210(A6)             Mrg Cop
fe8cd4 move.l  A3,A1                pass pointer to View in A1
fe8cd6 jsr     -222(A6)             Load View
fe8cda move.l  8(A5),A0             write pointer to View-Port A0
fe8cde lea -402(PC)(=$fe8b4e),A1    write colors to A1
fe8ce2 moveq   #$14,D0              pass Count in D0
fe8ce4 jsr     -192(A6)             Load RGB4
fe8ce8 move.l  16(A5),A3               pass pointer to Rast-Port A3
fe8cec lea     302(PC)(=$fe8e1c),A2 pass pass pointer to X1/Y1
                                       coordinates
fe8cf0 move.l  A3,A1                pass pointer to Rast-Port
                                    in A1
fe8cf2 moveq   #$00,D0              pass Draw Mode in D0
fe8cf4 jsr     -354(A6)             Set Draw Mode
fe8cf8 moveq   #$00,D3              clear D3
fe8cfa move.b  (A2)+,D3             pass Y1 coordinate in D3
fe8cfc moveq   #$00,D5              clear D5
fe8cfe move.b  (A2)+,D5             pass Pen in D5
fe8d00 cmpi.b  #$ff,D3              is Y1 equal to #$ff ?
fe8d04 bne.s   $fe8d2e              if not, branch
fe8d06 cmpi.b  #$ff,D5              is Pen equal to #$ff?
fe8d0a beq.l   $fe8d6a              if yes, branch
fe8d0e moveq   #$00,D4              clear D4
fe8d10 move.b  (A2)+,D4             get new X1 value
fe8d12 moveq   #$00,D3              clear D3
fe8d14 move.b  (A2)+,D3             get new Y1 value
fe8d16 move.l  A3,A1                write pointer to Rast-Port to A1

fe8d18 move.l  D5,D0                pass Pen in D0
fe8d1a jsr     -342(A6)             Set A Pen
fe8d1e moveq   #$28,D1              pass Y in D1
fe8d20 add.l   D3,D1                add Y1 to Y
fe8d22 moveq   #$46,D0              pass X in D0
fe8d24 add.l   D4,D0                add X1 to X

fe8d26 move.l  A3,A1                write pointer to Rast-Port to
fe8d28 jsr     -240(A6)             Move
```

```
fe8d2e cmpi.b  #$fe,D3             is Y1 equal to #$fe
fe8d32 bne.s   $fe8d56             if not, branch
fe8d34 moveq   #$00,D4             clear D4
fe8d36 move.b  (A2)+,D4            get new X1 value
fe8d38 moveq   #$00,D3             clear D3
fe8d3a move.b  (A2)+,D3            get new Y1 value
fe8d3c move.l  A3,A1               pass pointer to Rast-Port in A1
fe8d3e move.l  D5,D0               pass Pen in D0
fe8d40 jsr     -342(A6)            Set A Pen
fe8d44 moveq   #$28,D1             pass Y in D1
fe8d46 add.l   D3,D1               add Y1 to Y
fe8d48 moveq   #$46,D0             pass X in D0
fe8d4a add.l   D4,D0               add X1 to X
fe8d4c moveq   #$01,D2             write Mode in D2
fe8d4e move.l  A3,A1               pass pointer to Rast-Port in A1
fe8d50 jsr     -330(A6)            Flood
fe8d54 bra.s   $fe8cf8             branch
fe8d56 move.l  D3,D4               pass Y1 in X1
fe8d58 move.l  D5,D3               pass Pen in Y1
fe8d5a moveq   #$28,D1             write Y to D1
fe8d5c add.l   D3,D1               add Y1 to Y
fe8d5e moveq   #$46,D0             write X to D0
fe8d60 add.l   D4,D0               add X1 to X
fe8d62 move.l  A3,A1               pointer to Rast-Port to A1
fe8d64 jsr     -246(A6)            Draw
fe8d68 bra.s   $fe8cf8             branch
fe8d6a lea     588(PC)(=$fe8fb8),A2 pointer to new data
fe8d6e move.l  A3,A1               pass pointer to Rast-Port in A1
fe8d70 moveq   #$03,D0             pass Pen in D0
fe8d72 jsr     -342(A6)            Set A Pen
fe8d76 move.w  (A2)+,D0            get Pen
fe8d78 bmi.s   $fe8db8             if result is negative, branch
fe8d7a move.b  D0,24(A3)           store Pen in Rast-Port
fe8d7e moveq   #$00,D4             clear D4
fe8d80 move.b  (A2)+,D4            get X1
fe8d82 moveq   #$00,D5             clear D5
fe8d84 move.b  (A2)+,D5            get Pen
fe8d86 moveq   #$46,D2             load D2 with #$46
fe8d88 moveq   #$00,D0             clear D0
fe8d8a move.b  (A2)+,D0            get D0
fe8d8c add.l   D0,D2               add D0 to D2
fe8d8e moveq   #$28,D3             get Y1
fe8d90 move.b  (A2)+,D0            get Y
fe8d92 add.l   D0,D3               add Y to Y1
fe8d94 move.w  D4,D0               pass X1 in D0
fe8d96 mulu    D5,D0               multiply X1 with D5
fe8d98 lea     1032(A4),A0         pass Source in A0
fe8d9c bra.s   $fe8da0             branch
```

The next routine copies display data into the Source structure.

```
fe8d9e move.w  (A2)+,(A0)+      copy data in the Source structure
fe8da0 dbf     D0,$fe8d9e       if not all data were copied, branch
fe8da4 lea     1032(A4),A0      pass Source in A0
fe8da8 moveq   #$00,D0          pass srcX in D0
fe8daa add.l   D4,D4            increment sizeX
fe8dac move.l  D4,D1            pass srcMod in D1
fe8dae move.l  A3,A1            store destRast-Port
fe8db0 lsl.w   #3,D4            store sizeX
fe8db2 jsr     -36(A6)          Blt Tem Plate
fe8db6 bra.s   $fe8d76          branch
```

The next routine passes the colors in the View port colormap.

```
fe8db8 move.l  8(A5),A0                 pass pointer to View-Port
                                        in A0
fe8dbc lea     -584(PC)(=$fe8b76),A1    write pointer to colors in
                                        A1
fe8dc0 moveq   #$04,D0                  pass Count in D0
fe8dc2 jsr     -192(A6)                 Load RGB4
fe8dc6 jsr     -270(A6)                 Wait T Of
fe8dca movem.l (A7)+,A6/A3-A2/D5-D2     restore Register
fe8dce rts
```

The next routine releases the View port and closes the Gfx library.

```
fe8dd0 move.l  A6,-(A7)           save Register
fe8dd2 tst.l   4(A5)              Gfx-Base available ?
fe8dd6 beq.s   $fe8e16            if not, branch
fe8dd8 tst.l   8(A5)              pointer to View-Port
                                  present ?
fe8ddc beq.s   $fe8e0a            if not, branch
fe8dde move.w  #$0100,$dff096     block DMA
fe8de6 move.l  4(A5),A6           pass Gfx-Base in A6
fe8dea suba.l  A1,A1              clear A1
fe8dec jsr     -222(A6)           Load View
fe8df0 move.l  8(A5),A0           pass pointer to View-Port in A0
fe8df4 jsr     -540(A6)           FreeVPort Cop Lists
fe8df8 move.l  0(A5),A6           pass Execbase in A6
fe8dfc move.l  8(A5),A1           write pointer to View-Port to
A1
fe8e00 move.l  #$00005e9a,D0      pass number of Bytes in D0
fe8e06 jsr     -210(A6)           Free Mem
fe8e0a move.l  0(A5),A6           pass Execbase in A6
fe8e0e move.l  4(A5),A1           pass Gfx-Base in A1
fe8e12 jsr     -414(A6)           Close Library
fe8e16 move.l  (A7)+,A6           get Register
fe8e18 rts
```

6.2.3 The root block

AmigaDOS controls all files through tables of content, the directories. Every directory can contain another. The difficulty is in the fact that AmigaDOS must partition the logical blocks (sectors) of the Trackdisk devices into directories and files. This is done with the help of control blocks. These blocks are only used for control and therefore do not contain any of the actual data.

The root block is one of these control blocks. It represents the main directory and contains, among other items, the name of the disk. The root block is located in cylinder 40, upper side sector zero. This is block 880 ($370).

LONG-Offset	Byte-Nr	Function	Constants
0	0	Type T SHORT	(2)
1	4	Always 0	(0)
2	8	Always 0	(0)
3	12	Size of Hashtable (size-56)	(72)
4	16	Always 0	(0)
5	20	Checksum	
6	24	Hash table	
Size-50	312	BMFLAG-> <>0: Bitmap is valid	
Size-49	316	Bitmap-Blocks	
Size-23	420	Last Write access: Day	
Size-22	424	Last Write access: Minutes	
Size-21	428	Last Write access: Ticks (1/50Sec)	
Size-20	432	Disk Name as BCPL-String (<=30 character)	
Size-7	484	Disk Creation date: Day	
Size-6	488	Disk Creation date: Minutes	
Size-5	492	Disk Creation date: Ticks (1/50Sec)	
Size-4	496	Always Null	(0)
Size-3	500	Always Null	(0)
Size-2	504	Always Null	(0)
Size-1	508	Sub type of the Block	ST.ROOT (1)

All entries are in the longword format as it is customary in BCPL. Since AmigaDOS does not prescribe the length of a logical block, the individual entries are always counted relative to the beginning and the end.

For disks a block length of 512K is fixed. The byte indications in the table have already been converted to the standard length of the disk sector. The longword indications count relative to the beginning or end, where size is the length of a block in longwords (for a Floppy 128L).

At the beginning of every block there is an identification which indicates the type of the block. In this case it is a 2 for T.SHORT. In addition at the end of the block there is a sub-identification. The root block contains a 1 for ST.ROOT.

Also the root block contains the name of the disk as a BCPL string. The date on which the disk was formatted can also be found here. AmigaDOS also makes a "note" here of the last time the disk was written.

The function of the remaining entries is described in the next sections.

6.2.4 The user directory blocks

This type of block controls the subdirectories. The main type is the same as the boot block. The subtype is again the block: ST.USERDIR.

The construction in general is the same as the Root directory:

LONG-Offset	Byte-Nr	Function	Constants
0	0	Type	T.SHORT (2)
1	4	Block-pointer to itself	
2	8	Always 0	(0)
3	12	Always 0	(0)
4	16	Always 0	(0)
5	20	Checksum	
6	24	Hashtable	
Size-50	312	Not used	(0)
Size-48	320	Protection Status-Bits	
Size-47	324	Not used	(0)
Size-46	328	Comment as BCPL-String	
Size-23	420	Creation date: Day	
Size-22	424	Creation date: Minutes	
Size-21	428	Creation date: Ticks (1/50Sec)	
Size-20	432	Dir name as BCPL string (<=30 char.)	
Size-4	496	Next Block with same Hash	
Size-3	500	Block-pointer to higher level directory.	
Size-2	504	Always Null	(0)
Size-1	508	Sub type of Block	ST.USERDIR (2)

6.2.5 The File header block

This block is the backbone of every file. The type is T.SHORT with the subtype ST.FILE. This block holds the pointers to the individual data blocks:

LONG-Offset	Byte-Nr.	Function	Constants
0	0	Type T	SHORT (2)
1	4	Block-pointer to itself	
2	8	Number of Blocks in File-Header(!)	
3	12	Always Null	(0)
4	16	First data block	
5	20	Checksum	
6	24	LAST Block-pointer to the Data Blocks	
Size-51	308	FIRST Block-pointer to data	
Size-50	312	Not used (0)	

Size-48	320	Protection Status-Bits
Size-47	324	Size of the Files in Bytes
Size-46	328	Comments as BCPL-String
Size-23	420	Creation date: Day
Size-22	424	Creation date: Minutes
Size-21	428	Creation date: Ticks (1/50Sec)
Size-20	432	Filename as BCPL-String (<=30 characters)
Size-4	496	Next Block with the same Hash
Size-3	500	Block-pointer to higher level Directory
Size-2	504	Block-pointer to first Extension-Block or 0, if all Blocks are recorded here
Size-1	508	Sub type of the Block ST.FILE (-3)

6.2.6 The File list block

If there is not enough space in the File header block for all the pointers, this block contains the remaining entries. If this block does not have enough space, another File list block is attached. This happens until all data blocks have been accomodated.

LONG-Offset	Byte-Nr.	Function	Constants
0	0	Type	T.LIST (16)
1	4	Block-pointer to itself	
2	8	Number of Blocks noted in File List(!)	
3	12	Always Null	(0)
4	16	First Data block	
5	20	Checksum	
6	24	Last Block pointer to the Data blocks.	
Size-51	308	First Block pointer to data	
Size-50	312	Not used	(0)
Size-4	496	always Null	(0)
Size-3	500	pointer to File-Header-Block	
Size-2	504	Block pointer to next Extension-Block or 0, if all Blocks are recorded	
Size-1	508	Sub type of the Block ST.FILE	(-3)

6.2.7 The Data block

The block numbers of the individual data blocks are contained in the File header or File list block. This is the data block which contains the actual bytes of a file:

LONG-Offset	Byte-Nr	Function	Constant
0	0	Type T.DATA	(8)
1	4	Pointer to File header block	
2	8	Number of the Data block	
3	12	Number of data bytes in the block	
4	16	Next Data block	
5	20	Checksum	
6	24	Data starts here	

6.2.8 The calculation of the checksum

As the tables show, the first longword (= 20K) in the block contains a checksum for the block. How this is calculated can be seen in the following small program.

```
        lea Databuffer,a0  ;pointer to Data Buffer
        move.l a0,a1       ;save pointer
        move.w #$7f,d1     ;Counter for number of data
        clr.l d0           ;clear D0
        move.l d0,20(a1)   ;clear sum entered
loop1:  sub.l (a0)+,d0
        dbf d1,loop1       ;form Sum
        move.l d0,20(a1)   ;enter Sum into Block
        rts
        END
```

6.3 Connections between the blocks

All blocks of a disk controlled by AmigaDOS are logically connected with each other. The root of this block system is the *root block*. It is always located in logical block 880 of the disk. From this block branches all other blocks.

The location of individual blocks is determined by the Hash Table. It contains (for a block length of 512K), 72 longword pointers. AmigaDOS calculates the proper entry in the Hash Table from the filename. It then checks in the block to determine if it was the desired entry. More on this in the next section. Assume that the Hash Table contains the pointers to the control blocks of the files and sub-directories.

The following illustration shows the complete file system of AmigaDOS with all control blocks:

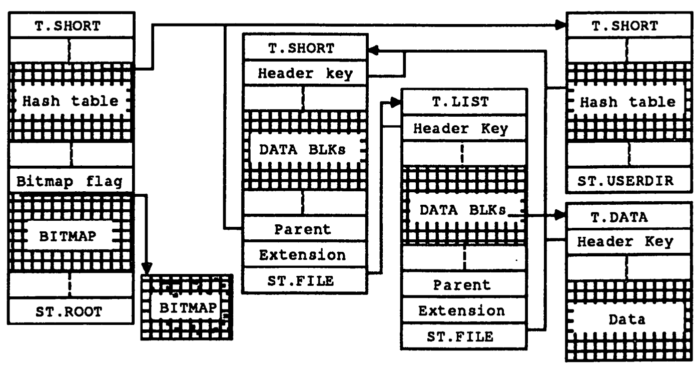


Figure 1: The Amiga File System

The root block (extreme left) also contains some pointers to the Bitmap blocks. In the Bitmap, AmigaDOS records which blocks are still available and which are occupied. A normal disk requires less than one block for the Bitmap. For this reason only the first Bitmap pointer of the root block is occupied.

The Hash Table of the root block can contain pointers to a sub-directory (ST.USERDIR) or to a file (ST.FILE). A sub-directory is constructed similar to the root block. The Hash Table has the same construction. It contains only the block pointers to the files and other sub-directories of ST.USERDIR's.

For a normal file the Hash entry points to a File header ST.FILE. Every File header contains the individual pointers to the data blocks. If there is not sufficient space for the pointers in the File header, the extension entry points to an extension block. This block is similar in construction to the file header and contains the remaining pointers to the data blocks. If even this space is not sufficient, the extension entry points to the next file list block until all data blocks have been recorded.

The data blocks contain only 6 longwords for control. An entry always points to the next data block of the same file. Additionally, the number of data bytes contained in this block are recorded here. In case of doubt this must always be the maximum of 488K (512-6*8). Only the last data block can be partially empty.

6.4 The hash calculation

The user familiar with other file systems automatically connects a directory with a list of filenames. In AmigaDOS, however, a directory is organized differently. The filenames are already recorded in the File header block. The directory is therefore only a list of block numbers in which the headers can be found.

The problem is to get from the filenames to the hash entry belonging to it. This is done with a form of "checksum calculation" of the Name string:

```
UBYTE Capital( c )UBYTE c;
{
   if(c >= 'a' && c <= 'z')
      c -= 'a'-'A';
   return c;
}
LONG Hash( length, s )LONG length;UBYTE *s;
{
LONG hash;
   for( hash = length; length--; )
      hash = ((hash*13 + Capital( *s++ )) & 0x7ff);
   return (LONG) (hash % 72 + 6 );
}
```

The hash value

The string and its length is required for the calculation of the hash value. The routine does not differentiate between upper and lowercase letters and basically calculates with the ASCII value of the uppercase letters. For this reason the Capital function is used for the conversion.

When the hash value has been computed in the loop, it can be a number up to 2,047. This value must now be converted to the actual size of the Hash table. In a 512K disk block, the Hash table has 72 entries. Therefore the hash value is the result of dividing by 72. Since the Hash table starts after the sixth longword, a six must be added to the calculated value. Voila! - the entry of the Hash table has been calculated.

The calculated longword of the root or userdir block must be read to obtain the desired block number.

The Hashchain

This form of hash calculation has only one error. Several filenames can have the same hash value (there are only 72 possibilities). For this reason every header block has a clever entry, the "Hashchain". The header blocks with the same hash value are thus chained together. If no further files are contained in the Hashchain, the entry contains a 0.

6.5 The bitmap

In the root block an entry points to the bitmap block of the disk. This block contains the assignment of the blocks. Every bit represents a block. If a bit is set, the block is available. If the bit is reset (0), the block is already occupied.

The bitmap starts at the second longword of the bitmap blocks, since the first longword is a checksum for the block. Therefore, the lowest bit of the second longword represents the second block, instead of the zero block as you might have thought. In AmigaDOS the two boot blocks are always occupied. For this reason they do not even appear.

Following is a bitmap analysis program. The ¶'s in the program are not to be enterd, they only show where the line actually ends. The program first reads the root block and then gets the block number of the bitmap which it reads and represents graphically. All Trackdisk device operations are used in extended format:

```
/*------------------------------------------------------*/¶
/*                  Bitmap - Analyzer                   */¶
/*                                                      */¶
/*                    JEA, 08-15-87                     */¶
/*------------------------------------------------------*/¶
#include <exec/exec.h>¶
#include <devices/trackdisk.h>¶
#include <intuition/intuition.h>¶
#define ON 1L¶
#define OFF 0L¶
#define BLOCK_SIZE 128L¶
#define BM_FLAG BLOCK_SIZE-50L¶
#define BM_BLOCKS BLOCK_SIZE-49L¶
extern struct MsgPort *CreatePort();¶
extern struct IORequest *CreateExtIO();¶
struct IntuitionBase *IntuitionBase;¶
struct GfxBase *GfxBase;¶
struct TextAttr MyFont =¶
{¶
   "topaz.font",¶
   TOPAZ_EIGHTY,¶
   FS_NORMAL,¶
   FPF_ROMFONT,¶
};¶
struct NewScreen NewScreen =¶
{¶
   0,¶
   0,¶
   640,¶
   200,¶
```

```
   2,¶
   0, 1,¶
   HIRES|SPRITES,¶
   CUSTOMSCREEN,¶
   &MyFont,¶
   "- BitMap -",¶
   NULL,¶
   NULL,¶
);¶
/*----------------------------------------------------*/¶
/*              Switch Motor on and off               */¶
/*                                                    */¶
/*                                                    */¶
/*----------------------------------------------------*/¶
Motor( diskreq, on )¶
struct IOExtTD *diskreq;¶
LONG on;¶
{¶
   diskreq->iotd_Req.io_Length = on;¶
   diskreq->iotd_Req.io_Command = TD_MOTOR;¶
   DoIO(diskreq);¶
   return(0);¶
}¶
/*----------------------------------------------------*/¶
/*           Read Block from Device indicated         */¶
/*                                                    */
/*                                                    */¶
/*----------------------------------------------------*/¶
ReadBlock( diskreq, block, puffer, diskChangeCount )¶
struct IOExtTD *diskreq;¶
LONG block;¶
APTR puffer;¶
ULONG diskChangeCount;¶
{¶
   diskreq->iotd_Req.io_Length  = TD_SECTOR;¶
   diskreq->iotd_Req.io_Data    = puffer;¶
   diskreq->iotd_Req.io_Command = ETD_READ;¶
   diskreq->iotd_Count          = diskChangeCount;¶
   diskreq->iotd_Req.io_Offset  = block * TD_SECTOR;¶
   if(DoIO(diskreq))¶
      return(1);¶
   return(0);¶
}¶
/*----------------------------------------------------*/¶
/*            Search for Bitmap-Block and read        */¶
/*                                                    */¶
/*                                                    */¶
/*----------------------------------------------------*/¶
LONG¶
ReadBitmap( diskreq, buf )¶
struct IOExtTD *diskreq;¶
LONG *buf;¶
{¶
ULONG diskChangeCount;¶
   diskreq->iotd_Req.io_Command = TD_CHANGENUM;¶
   DoIO(diskreq);¶
```

```
   diskChangeCount = diskreq->iotd_Req.io_Actual;¶
   Motor( diskreq, ON );¶
   ReadBlock( diskreq, 880L, buf, diskChangeCount );¶
   printf¶
     ("Flag: %ld ,  Block#%ld \n", buf[BM_FLAG],
buf[BM_BLOCKS] );¶
   if( buf[BM_FLAG] )¶
      ReadBlock( diskreq, buf[BM_BLOCKS], buf );¶
   Motor( diskreq, OFF );¶
   return( buf[BM_FLAG] );¶
}¶
/*----------------------------------------------------*/¶
/*         Display Bitmap of the Device indicated     */¶
/*                                                    */¶
/*                                                    */¶
/*----------------------------------------------------*/¶
DisplayBitmap( diskreq, Screen )¶
struct IOExtTD *diskreq;¶
struct Screen *Screen;¶
{¶
LONG *buf;¶
LONG x, y;¶
ULONG loop;¶
struct Window *Window;¶
struct NewWindow NewWindow;¶
ULONG MessageClass;¶
USHORT code;¶
LONG flag;¶
struct Message *GetMsg();¶
struct IntuiMessage *message;¶
   NewWindow.LeftEdge = 0;¶
   NewWindow.TopEdge = 0;¶
   NewWindow.Width = 640;¶
   NewWindow.Height = 160;¶
   NewWindow.DetailPen = 0;¶
   NewWindow.BlockPen = 1;¶
   NewWindow.Title = " Bitmap ";¶
   NewWindow.Flags = WINDOWCLOSE|SMART_REFRESH|ACTIVATE|¶
                     WINDOWDRAG|WINDOWDEPTH|¶
                     NOCAREREFRESH | GIMMEZEROZERO;¶
   NewWindow.IDCMPFlags =
CLOSEWINDOW|DISKINSERTED|DISKREMOVED;¶
   NewWindow.Type = CUSTOMSCREEN;¶
   NewWindow.FirstGadget = NULL;¶
   NewWindow.CheckMark = NULL;¶
   NewWindow.Screen = Screen;¶
   NewWindow.BitMap = NULL;¶
   NewWindow.MinWidth = 640;¶
   NewWindow.MinHeight = 148;¶
   NewWindow.MaxWidth = 640;¶
   NewWindow.MaxHeight = 200;¶
   if(( Window = (struct Window*) ¶
                     OpenWindow( &NewWindow )) == NULL)¶
     exit( FALSE );¶
   SetAPen (Window->RPort, 1 );¶
   Move( Window->RPort, 500, 33 );¶
```

```
    Text( Window->RPort, "Side  0", 7 );¶
    Move( Window->RPort, 500, 105 );¶
    Text( Window->RPort, "Side  1", 7 );¶
    Move( Window->RPort, 0, 146 );¶
    Text( Window->RPort, "1", 1 );¶
    Move( Window->RPort, 466, 146 );¶
    Text( Window->RPort, "80", 2 );¶
    Move( Window->RPort, 480, 8 );¶
    Text( Window->RPort, "Sector 1.", 9 );¶
    Move( Window->RPort, 480, 64 );¶
    Text( Window->RPort, "Sector 11.", 10 );¶
    Move( Window->RPort, 480, 80 );¶
    Text( Window->RPort, "Sector 1.", 9 );¶
    Move( Window->RPort, 480, 136 );¶
    Text( Window->RPort, "Sector 11.", 10 );¶
    buf = (LONG*) AllocMem( 512L, MEMF_CHIP );¶
    ReadBitmap( diskreq, buf );¶
    buf[0] &= 0x3fffffffL;¶
    loop=30L;¶
    for( x=0; x<80; ++x ){¶
       for( y=0; y<22; ++y ){¶
          if( buf[loop/32] & (1L << (loop % 32)) )¶
             SetAPen (Window->RPort, 2 );¶
          else¶
             SetAPen (Window->RPort, 3 );¶
          if (y > 10)¶
             RectFill( Window->RPort, x*6, ¶
                              (y+1)*6, x*6+4, (y+1)*6+4 );¶
          else¶
             RectFill( Window->RPort, x*6, y*6, x*6+4,
y*6+4 );          ++loop;¶
       }¶
    }¶
    FreeMem( buf, 512L );¶
    Wait( 1<<Window->UserPort->mp_SigBit);¶
     flag = TRUE;¶
     do¶
     {¶
        if (message = (struct ¶
              IntuiMessage *)GetMsg(Window->UserPort))  {¶
          MessageClass = message->Class;¶
          code = message->Code;¶
          ReplyMsg(message);¶
          switch (MessageClass) {¶
¶
             case CLOSEWINDOW : flag = FALSE;¶
                                break;¶
             case DISKREMOVED :¶
                                Text( Window->RPort, ¶
                                     "Disk Removed", 11);¶
                                break;¶
          }   /* Case */¶
       }  /* if */¶
     }¶
    while( flag );¶
    CloseWindow( Window );¶
```

```
}¶
/*--------------------------------------------------------*/¶
/*                   Close Libraries                       */¶
/*                                                         */¶
/*                                                         */¶
/*--------------------------------------------------------*/¶
CloseLibs()¶
{¶
   CloseLibrary( IntuitionBase );¶
   CloseLibrary( GfxBase );¶
}¶
/*--------------------------------------------------------*/¶
/*                      Open Libraries                     */¶
/*                                                         */¶
/*                                                         */¶
/*--------------------------------------------------------*/¶
LONG¶
OpenLibs()¶
¶
{¶
   IntuitionBase = (struct IntuitionBase*)¶
                   OpenLibrary("intuition.library", 0);¶
   if ( IntuitionBase == NULL ) exit( FALSE );¶
   GfxBase       = (struct GfxBase*)¶
                   OpenLibrary("graphics.library", 0);¶
   if (GfxBase == NULL) exit ( FALSE );¶
   return( TRUE );¶
}¶
/*--------------------------------------------------------*/¶
/*              Main-Program BITMAP-Analyzer               */¶
/*                                                         */¶
/*                (open Device and Screen)                 */¶
/*--------------------------------------------------------*/¶
main()¶
{¶
struct MsgPort *diskport;¶
struct IOExtTD *diskreq;¶
struct Screen *Screen;¶
   if (!OpenLibs()) exit(FALSE);¶
   if ((diskport = CreatePort(0L,0)) == NULL)¶
   {¶
      printf("Port can't be opened\n");¶
      exit(FALSE);¶
   }¶
   diskreq = (struct IOExtTD *)CreateExtIO(diskport,¶
                        (long)sizeof(struct IOExtTD));¶
   if (diskreq == 0)¶
   {¶
      printf("DiskRequest can't be created !, Error
%ld\n",diskreq);¶
¶
      DeletePort(diskport);¶
      exit(FALSE);¶
   }¶
   if ( OpenDevice(TD_NAME, 0L, diskreq, 0L) )¶
   {¶
```

```
        printf("Device reports error!\n");¶
        DeletePort(diskport);¶
        DeleteExtIO(diskreq,(long)sizeof(struct IOExtTD));¶
        exit(FALSE);¶
    }¶
  ¶
    if( (Screen = (struct Screen*)OpenScreen(&NewScreen))
== NULL )¶
      exit( FALSE );¶
    DisplayBitmap( diskreq, Screen );¶
    DeleteExtIO(diskreq,(long)sizeof(struct IOExtTD));¶
    DeletePort(diskport);¶
    CloseScreen( Screen );¶
    CloseLibs();¶
    exit( TRUE );¶
}¶
```

Calculation of the bitmap checksum

The checksum of "normal" blocks differs from that of the bitmap block only by the position where it is entered into the block. In the bitmap block it is the first longword. The following program calculates the checksum and stores it.

```
          lea Databuffer,a0    ;pointer to Data Buffer
           move.l a0,a1        ;save pointer
           move.w #$7f,d1      ;Counter for number of data
           clr.l d0            ;clear D0
           move.l d0,(a1)       ;erase sum entered
loop1:     sub.l (a0)+,d0
           dbf d1,loop1        ;form Sum
           move.l d0,(a1)      ;enter Sum into Block
           rts
           END
```

7.
Viruses

7. Viruses

Almost everybody has heard about the existence of computer viruses. How they are constructed and protection against them, is probably not well known.

Generally computer viruses are small programs which integrate themselves into the operating system of a computer. At the most favorable opportunity they copy (multiply) themselves onto diskettes or hard disks.

Besides having the capability of reproduction, the viruses often cause unpleasant effects inside the computer. This can start with a harmless message on the screen. Unfortunately, it can escalate to a crash of the system or the destruction of the data on disks.

A virus is, as already mentioned, a program and must therefore be started like any other program to become active. To start itself without the user noticing, the virus copies itself to a spot on the disk where it is not noticeable and is started automatically.

7.1 Boot block viruses

The simplest place for the virus to go is the boot block, where normally important data isn't stored. It's large enough to host a virus and the virus is started after a reset when a disk is inserted into the internal drive of the Amiga. On the basis of this fact, the first viruses which afflicted the Amiga, were stored in the boot block.

To create an effective virus, it must always be protected from being deleted from memory when a reset occurs. The reset routine can be altered so that it can be used to branch into the virus program and to start the reproduction process.

The SCA virus

The first virus which used the boot block is the well known SCA (Swiss Cracker's Association) virus. It is of course reset protected. It also uses the disk reset to reproduce itself. For reproduction the virus is written into the boot block of the disk which is in the drive during the disk reset. After a certain number of copy processes, the virus displays a message on the screen.

Since this virus was one of the first of its species it spread widely because few users knew how to find and remove it.

SCA made few friends with this virus which contained a bragging message. It was possible for it to destroy the important copy protection information on a commercial disk, therefore making the disk unusable.

The Byte Bandit virus

This SCA virus was followed by other boot block viruses. One of these is the Byte Bandit virus which is programmed more elegantly, but has some errors.

It uses the fact that the boot block is read after every insertion of a disk to verify the validity of the disk. All read and write procedures which are initiated by the operating system on the disk, are performed through the trackdisk device. A message is sent to the device and is processed after a jump through a vector contained in the device structure. The virus changes this vector into its own program. It intercepts the message which indicates that the boot block should be read, and changes the read into a Write command. Then the buffer pointer which was set for the read of the boot block, is set to the buffer containing the virus. This trick spreads the virus during the insertion of a disk which isn't write protected.

Since the procedure for removal of viruses was known by the time this virus appeared, it did not become as widespread as the SCA virus. This was lucky since the virus was not very harmless. It caused a system crash which could only be remedied with a reset unless you know more about the virus (more on this later). The idea (not very original) of causing a system crash was probably selected because the boot block did not offer more storage space.

7.2 Virus rumors

Rumors have circulated that viruses exist which could place themselves inside the battery backed memory of the real time clock of the Amiga 2000 and Amiga 500. These viruses would then remain active even after the computer was switched off. These reports appeared in several publications. This was surprising since the authors (undoubtedly not computer novices) should have been aware of the fact that this is absolutely impossible. Impossible, since the battery backed memory is not large enopugh to store a virus. Even if it was sufficient (the clock buffer is about 1K) to be stored in the computer after the power was switched off, the vector to the virus would have disappeared. A program in memory which is not started does not have the capability to spread itself and the operating system does not execute the data in the clock as a program. Therefore, this virus variant is a non-executable idea.

Another rumor was spread that a virus existed which integrated itself into the write protected RAM of the Amiga 1000 where the operating system was stored.

It's unlikely that a virus of this type exists, but under certain circumstances a program can be stored in the Kickstart area of the Amiga 1000.

Writing a program, even a virus, into the Kickstart area is only possible when a RAM expansion is present. In this case the Kickstart area can be made ready for writing with the assembler command Reset without switching off the RAM area where the program is stored (but only the Fast RAM). If the Reset command is executed in the Chip RAM, it switches off its own RAM, and this leads to an unavoidable system crash.

Another virus type has surfaced recently. These are the viruses which insert themselves into the `CLI` commands (such as the Dir command or the disk validator.) An attempt to execute an infected `CLI` command, or a disk validation, makes the virus active. Until now no such virus has been observed on the Amiga by the author. It is uncertain that they really exist, but it is probably possible to program a virus of this type.

The dangerous aspect of this last type of virus is the fact that it is not limited to disks. It could spread to hard disk units. In addition it would be harder to find than the boot virus.

Viruses which insert themselves into existing programs can only work on two principles. Either they enlarge the code by copying themselves behind the program, or they destroy the actual program code so no additional space is used on the disk. In both cases the virus can be easily identified and dealt with because of this.

7.3 Protection against viruses

A general protection scheme against viruses unfortunately cannot be provided. Their spread can be limited by using only write protected disks whenever possible.

The removal of boot block viruses is in general not a very difficult problem. The boot block can be initalized again with the CLI command Install. Please note that the computer must be switched off after the appearance of the virus and rebooted with a disk which is guaranteed not to be infected. This is very important, because viruses whose construction correspond to the Byte Bandit virus intercept the Install command and write the virus on the disk again. To remove the virus from the disks, the user must ensure that the virus is not in the computer.

The sequence for removing a Boot block virus is as follows:

1.) Switch the computer off.

2.) Boot with a disk which is not infected.

3.) Copy the Install command onto the RAM disk:

```
copy sys:c/install ram:
```

4.) Remove the CLI disk from DF0 and insert the infected disk.

5.) Erase virus from disk:

```
ram:install df0:
```

These five steps remove the virus from an infected disk. Caution should be used in using this technique since the mere fact that a disk has a non-standard boot block doesn't mean that it is infected, especially if it is a commercial program.

Finally a useful tip. If a computer should crash without a recognizable reason (the display disappears and the computer doesn't react to anything) this could be caused by the Byte Bandit virus. In this case the computer should not be reset which would cause the loss of data. Pressing the lowest five keys from left to right, at the same time, brings the computer to "life" again. The virus is still in the computer, but the possibility exists to store the working files.

The key combination is:

```
[ALT]  [Commodre]  [SPACE]  [AMIGA]  [ALT]
```

8.
The Trackdisk device

8. The Trackdisk device

We began the AmigaDOS chapter with a discussion of the partition hierarchy of the operating system. As we've seen the top level is AmigaDOS itself. The next level is the Trackdisk device.

AmigaDOS uses the Trackdisk device for all disk operations. It has a fairly limited set of commands, but forms the heart of every disk access. It determines whether a disk is in the drive and if so, whether it is write protected. The most important assignment of the Trackdisk device is the reading and writing of information on the disk.

Before discussing the usage of the Trackdisk device, you must understand how data is stored on disk.

8.1 Divisions of a disk

Stepper motor

A disk–regardless of its size–is roughly divided into two structures. The first structure consists of concentric rings called *tracks*. Since the Amiga disk has two sides, the two tracks which lie on top of each other on either side comprise a cylinder. The read/write heads are always positioned over one of these cylinders. The heads are moved from one cylinder to another by the *stepper motor*.

A normal Amiga disk consists of 80 cylinders with the motor moving the read/write heads between them. Cylinder 0 is located on the outermost ring and cylinder 79 on the innermost. Since there are two tracks per cylinder you end up with 160 tracks. The tracks on the lower side of the disk have even numbers, those on the upper side have odd numbers. Track 0 is in cylinder 0 on the upper side; track 1 on the lower side of cylinder 0. It follows that the last track (track 159) can be found on cylinder 79 of the lower side.

A track is divided into several sectors. The sectors are located consecutively on a track. The Amiga has 11 of these sectors in one track. The sector contains the actual data, 512 bytes.

With these facts the storage capacity of a disk can be calculated. A disk has two sides. Each one of these sides has 80 tracks. Each track has 11 sectors. Each sector contains 512 bytes. This results in:

```
80 Cylinder * 2 Tracks * 11 Sectors * 512 bytes= 901,120 bytes
   or (Bytes/1024=K) = 880K
```

Blocks

The Trackdisk device does not calculate in side/track/sector format, but in logical sectors, called *blocks*. A block always corresponds to a sector somewhere on the disk. The blocks are numbered sequentially from 0 to 1,759. This division makes control of the disk much easier since working with one variable (block number) is easier then with three; (side, track, sector).

The first 11 blocks are on Side A (upper disk side) in track and cylinder 0. The next 11 blocks (10-21) are on the lower side, also in cylinder 0, but in track 1.

This constant changing of sides might appear at first to be clumsy. Considering that both read/write heads are operated by one motor, this organization actually saves time. Very often a series of connected blocks must be read. The two heads move only once to read two tracks instead of once for each side.

The conversion of side track sector format to blocks uses the following formula: `Block = 2*11*Cylinder + 11*Side + Sector`

8.2 Devices and their applications

There are many different devices in the operating system of the Amiga, but all are constructed according to the same system.

A device always consists of a data structure (the Device structure) and a task (simultaneously running program), which accepts the commands of the user program. Sending a command to a device is basically no different from sending a command to another task which is recognized there and processed.

The complete transmission of the commands and results between tasks occurs through the message system of Exec. For this reason every task which wants to accept messages must provide a message port. The message transmission is similar to the transmission of a phone conversation. Without a telephone (message port), no conversation (message).

To send a command to the Trackdisk device (the Trackdisk task), a message port is created. This is necessary because the task must not only send commands, but must also be in the position to receive replies (reply messages).

A message port is a structure which in C appears as follows:

```
struct MsgPort
{                                       /* Offsets  */
struct     Node       mp_Node;          /* 0    $00 */
UBYTE      mp_Flags;                    /* 14   $0E */
UBYTE      mp_SigBit;                   /* 15   $0F */
struct     Task      *mp_SigTask;       /* 16   $10 */
struct     List       mp_MsgList;       /* 20   $14 */
};
```

The Node structure at the beginning connects the individual ports in a global list of the operating system. If the port should remain local, this structure remains unused.

A port is initialized with the Library function CreatePort. This function is normally already in the C library and does not have to be input.

```
#include        "exec/ports.h"
#include        "exec/memory.h"

struct MsgPort
*CreatePort(name, pri)
char *name;
long pri;
```

```
{
register struct MsgPort *mp;
register long sig;
long AllocSignal();
void *AllocMem();
struct Task *FindTask();

   if ((sig = AllocSignal(-1L)) == -1)
      return(0);
   if ((mp = AllocMem((long)sizeof(*mp), MEMF_PUBLIC
                                           |MEMF_CLEAR)) == 0)
    {
       FreeSignal(sig);
       return(0);
    }
    mp->mp_Node.ln_Name = name;
    mp->mp_Node.ln_Pri = pri;
    mp->mp_Node.ln_Type = NT_MSGPORT;
    mp->mp_Flags = 0;
    mp->mp_SigBit = sig;
    mp->mp_SigTask = FindTask(0L);
    if (name)
       AddPort(mp);
    else
      NewList(&mp->mp_MsgList);
    return(mp);
 }
```

The only parameters needed for this function are a Name string and a Priority. A name is only required for global message ports to make the search easier for other tasks. For device ports "NULL" is sufficient as a name. The priority does not matter and a null is also sufficient here as a parameter.

The function obtains some memory for the Message Port structure and initializes the most important entries.

First the Node structure is processed; then the pointer to the task whose message port is involved; and finally the function must set the Signalbit. The Signalbit tells the task later if a message has arrived at the port. Every task has only a limited number of Signalbits. One of the bits can be reserved with AllocSignal. For device accesses, the signal bit tells if the device is finished.

Finally the function adds a global port with AddPort to the other global ports. For local ports only a List structure for the messages to follow is initialized. This occurs again with a Library function: NewList.

If the port is no longer needed, it can be made to disappear with the DeletePort function:

```
#include        "exec/ports.h"
#include        "exec/memory.h"
```

```
DeletePort(mp)
register struct MsgPort *mp;
{
   if (mp->mp_Node.ln_Name)
      RemPort(mp);
   mp->mp_Node.ln_Type = -1;
   mp->mp_MsgList.lh_Head = -1;
   FreeSignal((long)mp->mp_SigBit);
   FreeMem(mp, (long)sizeof(*mp));
}
```

First DeletePort tests if this is a global port. If this is the case, the Exec function RemPort removes the message port from the message port list. Then the function releases the Signalbit and the memory which were occupied.

To send a command, an IORequest structure is needed. The backbone of an IORequest structure is a Message structure. The Message structure serves as an aid for the transmission of commands. The command to be transmitted is not in the Message structure, but in the higher level IORequest structure.

The Message structure has the following appearance:

```
struct Message
{                                            /* Offsets    */
struct    Node       mn_Node;                /*  0  $00    */
struct    MsgPort   *mn_ReplyPort;           /* 14  $0E    */
          UWORD      mn_Length;              /* 18  $12    */
};
```

The Node structure takes over the concatenation of the messages within the message port list. The ReplyPort is a pointer to the Message Port structure of the task to which the message is returned by the Trackdisk task with the return values at the end of the command. At the end is the length of the message in bytes. This indication is necessary since the actual message follows the Message structure and its length can vary. The message to be sent in this case is the IORequest structure.

The IORequest structure has the following appearance:

```
struct IORequest
{                                            /* Offsets    */
struct    Message    io_Message;             /*  0  $00    */
struct    Device    *io_Device;              /* 20  $14    */
struct    Unit      *io_Unit;                /* 24  $18    */
          UWORD      io_Command;             /* 28  $1C    */
          UBYTE      io_Flags;               /* 30  $1E    */
          BYTE       io_Error;               /* 31  $1F    */
};
```

The normal IORequest is not usable for the Trackdisk device and for this reason an extended version exists:

IOStandard Request:

```
struct IOStdReq
{                                              /* Offsets   */
struct     Message    io_Message;    /*  0  $00   */
struct     Device    *io_Device;     /* 20  $14   */
struct     Unit      *io_Unit;       /* 24  $18   */
           UWORD      io_Command;    /* 28  $1C   */
           UBYTE      io_Flags;      /* 30  $1E   */
           BYTE       io_Error;      /* 31  $1F   */
           ULONG      io_Actual;     /* 32  $20   */
           ULONG      io_Length;     /* 36  $24   */
           APTR       io_Data;       /* 40  $28   */
           ULONG      io_Offset;     /* 44  $2C   */
};
```

The complete device parameter interchange runs through the last seven entries of the IOStdReq structure. "io_Command" contains the code of the command to be executed. The other parameters result from the various commands.

To arrange an IORequest, there is a Library function:

```
struct IORequest
*CreateExtIO( mp, size )
struct MsgPort *mp;
long size;
{
register struct IORequest *iop;
void *AllocMem();

   if (mp == 0)
      return(0);
   if ((iop=AllocMem(size, MEMF_PUBLIC|MEMF_CLEAR)) == 0)
       return(0);
    iop->io_Message.mn_Node.ln_Type = NT_MESSAGE;
    iop->io_Message.mn_Length = size;
    iop->io_Message.mn_ReplyPort = mp;
    return(iop);
 }
```

This function obtains the required memory for the structure to be created. Since the length of the structure can vary, the length must be indicated in bytes. In this case the length is 48 bytes (length of the IOStdReq structure). The function also requires the address of the message ports to which the message is returned after the completion of the I/O command. Finally the call returns the address of the just created IORequest.

If the IORequest is no longer required, the occupied memory space should be released. This can be done with the following function:

```
DeleteExtIO(iop)
register struct IORequest *iop;
{
   if (iop == 0)
```

```
DeleteExtIO(iop)
register struct IORequest *iop;
{
   if (iop == 0)
      return;
   iop->io_Message.mn_Node.ln_Type = -1;
   iop->io_Device = -1;
   iop->io_Unit = -1;
   FreeMem(iop, (long)iop->io_Message.mn_Length);
}
```

Some effort can be saved by maintaining the IOStdReq structure. For this two small Library functions exist which are similar to the last one mentioned:

```
#include        <exec/io.h>

struct IOStdReq
*CreateStdIO(mp)
struct MsgPort *mp;
{
struct IOStdReq *CreateExtIO();

 return(CreateExtIO(mp, (long)sizeof(struct IOStdReq))); }

DeleteStdIO(iop)
struct IOStdReq *iop;
{
   DeleteExtIO(iop);
}
```

Nothing much new has been added with these functions. CreateStdIO only saves the indicate length of IOStdReq. For DeleteStdIO nothing changes.

8.3 Sending commands

The practical part of the chapter begins here. Additional information about the use of various structures will also be presented here.

Before work can start with a device, two structures must be initialized. Through CreatePort a message port is created for the replies from the Trackdisk devices. CreateStdIO produces a IOStdReq structure.

After this has happened, the program must prepare the device for access by the task. This happens with the OpenDevice() function. The return is the number of errors that may have occurred.

```
error = OpenDevice(devName, unitNumber, IORequest, flags)
  D0                  A0        D0          A1        D1
```

Name A pointer to a string, in which the name of the device has been stored. Here "trackdisk.device".

Unit The number of the unit which is accessed (0 to 3). For every attached drive there is a unique task. Through the unit number the message port of the task which is responsible for the drive is determined. The port is entered into the IORequest structure as io_Device pointer. The IORequest structure (the command) is then sent to this port.

IORequest The pointer to the IORequest structure which is sent to the message port of the Trackdisk task should be stored.

Flags Set to null for opening the Trackdisk devices.

Error The message returned from the OpenDevice function. A value not equal to null signals and error.

The opening of the devices appears as follows:

```
diskport = CreatePort( 0, 0 );
diskreq = CreateStdIO( diskport );
OpenDevice( TD_NAME, 0, diskreq, 0 );

...(Device accesses start here)...

CloseDevice( diskreq );
DeleteStdIO( diskreq );
DeletePort( diskport );
```

If the device was opened with OpenDevice, the command transmission can start. The figure below shows a device command's execution.

The transmission of the command in the form of a IORequest structure can be performed with two commands. Either DoIO or SendIO can be used, although the latter is not as common. The difference between the two functions is in the treatment of the message which was sent, after the Trackdisk task has received it.

The DoIO function waits until the Trackdisk task has completed its assignment and has returned a message. In contrast, SendIO does not wait for the return message from the task. This must be done by the programmer. Those not skilled in programming of devices, should only work with DoIO. A pointer to the IORequest structure is passed to the DoIO and SendIO functions.

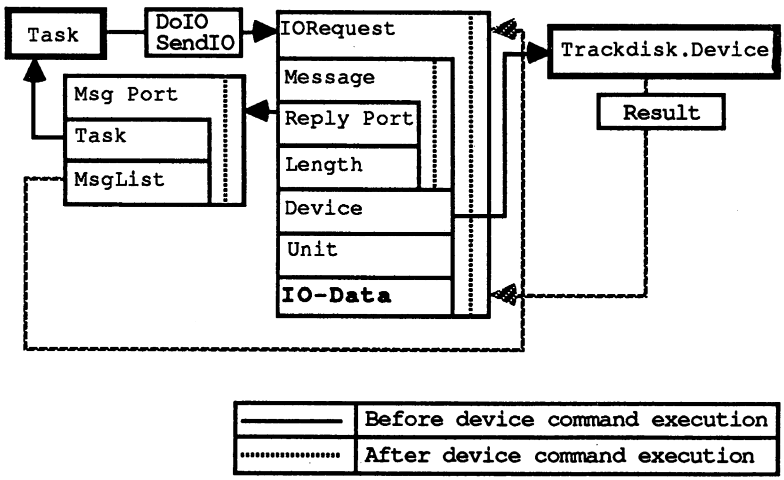


Figure 2: The execution of a Device command

The device gets a command code and possibly some parameters from IORequest (in the illustration under IO-Data).

Once the command has been executed (for example a sector has been read), or has been interrupted with AbortIO, the device sends the IORequest with the output parameters back to the reply port (port initialized by the task), which was indicated in the IORequest. This is done by the Exec function ReplyMsg. The user should not forget that every IORequest–no matter what type–is basically only a simple message!

If the message (the IORequest) arrives at the reply port, the Signalbit is released by Exec. If the task was waiting for the signal (during DoIO), it can now evaluate the result.

Here is a small program which tests the write protect of the disk in drive DF0:

```
/*--------------------------------------------------------*/
/*       Test Write Protect of the Diskette in DF0:      */
/*       use +L compile option, -lm -lc link option      */
/*                   JEA, 11-07-87                       */
/*--------------------------------------------------------*/
#include <exec/types.h>
#include <devices/trackdisk.h>

/*--------------------------------------------------------*/
/*                  Main Program                          */
/*                                                        */
/*                                                        */
/*--------------------------------------------------------*/
main()
{
struct MsgPort *diskport;
struct IOReq *diskreq;

  diskport = CreatePort( 0L, 0L );
  diskreq = CreateStdIO( diskport );
  OpenDevice( TD_NAME, 0L, diskreq, 0L );

  diskreq->io_Command = TD_PROTSTATUS;
  DoIO( diskreq );
  printf( "Write Protect: %ld\n", diskreq->io_Actual );

  CloseDevice( diskreq );
  DeleteStdIO( diskreq );
  DeletePort( diskport );
}
```

First the program opens the Trackdisk device with Unit 0 (DF0:). Then it passes the TD_PROTSTATUS command in the io_Command field of the IORequests.

The input parameters are now initialized. Only the IORequest remains to be sent by the program to the Trackdisk device. The simplest method is to use the Exec function DoIO. Exec waits until the device is finished and returns to the program. Since during the processing of the IORequest structure the program did not have to perform other tasks, it (our task) can go into waiting.

After DoIO has returned, the output parameter is in the io_Actual field of the IORequest output:

```
255 = Disk is write protected
  0 = Disk is not write protected
```

Once the write protect status has been output, the program must release all the structures it used. The program can be terminated only after this has occurred.

The most important assignment of the Trackdisk devices is the reading and writing of data on disks:

```
/*-----------------------------------------------------*/
/*           Read Sector with Trackdisk-Device         */
/*                                                     */
/*                  JEA, 12-07-87                      */
/*-----------------------------------------------------*/
#include <exec/types.h>
#include <exec/memory.h>
#include <devices/trackdisk.h>

/*-----------------------------------------------------*/
/*                 Turn Motor on or off                */
/*                                                     */
/*                                                     */
/*-----------------------------------------------------*/
MotorSwitch( iosr, flag )
struct IOStdReq *iosr;
LONG flag;
{
  iosr->io_Command = TD_MOTOR;
   iosr->io_Length = flag;        /* 1=an  and  0=aus */
   DoIO(iosr);
}

/*-----------------------------------------------------*/
/*                 Read Logical Block                  */
/*                                                     */
/*                                                     */
/*-----------------------------------------------------*/
LONG *GetBlock( iosr, block, map )
struct IOStdReq *iosr;
LONG block;
LONG *map;
{
LONG *ret = NULL;

  iosr->io_Command = CMD_READ;
  iosr->io_Length = TD_SECTOR;
  iosr->io_Data = (APTR)map;
  iosr->io_Offset = TD_SECTOR * block;
  DoIO(iosr);

  return(ret);
}

/*-----------------------------------------------------*/
/*                      Read Sector
/*                                                     */
/*-----------------------------------------------------*/
LONG *GetTSH( iosr, track, sector, head, map )
struct IOStdReq *iosr;
LONG track;
LONG sector;
LONG head;
LONG *map;
{
LONG *ret = NULL;
```

```
    iosr->io_Command = CMD_READ;
    iosr->io_Length = TD_SECTOR;
    iosr->io_Data = (APTR)map;
    iosr->io_Offset = TD_SECTOR*(sector + NUMSECS*head
                              + NUMSECS*NUMHEADS*track );
    DoIO(iosr);

    return(ret);
}

/*-------------------------------------------------------*/
/*                    Main Program                       */
/*                                                       */
/*-------------------------------------------------------*/
main()
{
struct MsgPort *diskport;
struct IOStdReq *diskreq;
LONG *buf;
LONG loop;

    buf = (LONG*) AllocMem( 512L, MEMF_CHIP );

    diskport = CreatePort( 0L, 0L );
    diskreq = CreateStdIO( diskport );
    OpenDevice( TD_NAME, 0, diskreq, 0 );

    MotorSwitch( diskreq, 1L );

    GetBlock( diskreq, 0L, buf );
    for( loop=0; loop<128; loop++ ){
       printf( "%lx ", buf[loop] );
    }
    printf( "\n\n" );

    GetTSH( diskreq, 0L, 0L, 0L, buf );
    for( loop=0; loop<128; loop++ ){
       printf( "%lx ", buf[loop] );
    }
    printf( "\n" );

    MotorSwitch( diskreq, 0L );

    CloseDevice( diskreq );
    DeleteStdIO( diskreq );
    DeletePort( diskport );

    FreeMem( buf, 512L );
}
```

Before this program opens the Trackdisk device, it reserves 512K of memory for itself in chip memory. A sector is read into this buffer later. Since the Trackdisk device uses the blitter for decoding of track data, the sector buffer must be in the lower part (in the lower 512K) of memory in the chip memory area.

Next the TD_MOTOR command switches on the motor of the drive since it must run for the read/write access. The Trackdisk device waits for the flag which determines if the motor is switched on (1) or off (0). This is done in the io_Length field of IORequest. DoIO then handles the device.

The CMD_READ command reads bytes from the disk. The field io_Length justifies its name. It contains the number of bytes to be read. In this case, since one sector is read, it is 512K.

The Trackdisk device looks in io_Data for the address of the buffer into which the byte read is copied.

In order to read a sector it must be known what sector we want to read. This is recorded, in bytes, in io_Offset. The program then converts the information from bytes to block numbers. When this is done the command is executed with DoIO.

Sometimes it is important to address a sector through the side, track, sector format instead of the logical block number. The conversion formula which was described earlier is very useful here.

For example, this program reads the first sector, called the bootsector, with the two possible methods and outputs the content in hexadecimal numbers.

After read access, the program switches the device motor off again, closes the device and releases the sector buffer.

8.3.1 The commands in overview

Every device can be addressed with a standard set of commands which are defined in the Include file "Exec/io.h":

```
#define CMD_INVALID 0L
#define CMD_RESET 1L
#define CMD_READ 2L
#define CMD_WRITE 3L
#define CMD_UPDATE 4L
#define CMD_CLEAR 5L
#define CMD_STOP 6L
#define CMD_START 7L
#define CMD_FLUSH 8L
#define CMD_NONSTD 9L
```

All of these standard commands begin with "CMD". Every device can, in addition, have its own special commands. These start at CMD_NONSTD. The additional commands of the Trackdisk devices are contained in the Include file "Devices/Trackdisk" offset to CMD_NONSTD:

```
#define TD_MOTOR (CMD_NONSTD+0)
#define TD_SEEK (CMD_NONSTD+1)
#define TD_FORMAT (CMD_NONSTD+2)
#define TD_REMOVE (CMD_NONSTD+3)
#define TD_CHANGENUM (CMD_NONSTD+4)
#define TD_CHANGESTATE (CMD_NONSTD+5)
#define TD_PROTSTATUS (CMD_NONSTD+6)
#define TD_RAWREAD (CMD_NONSTD+7)
#define TD_RAWWRITE (CMD_NONSTD+8)
#define TD_GETDRIVETYPE (CMD_NONSTD+9)
#define TD_GETNUMTRACKS (CMD_NONSTD+10)
#define TD_ADDCHANGEINT (CMD_NONSTD+11)
#define TD_REMCHANGEINT (CMD_NONSTD+12)
#define TD_LASTCOMM (CMD_NONSTD+13)

#define TDF_EXTCOM (1L<<15)

#define ETD_WRITE (CMD_WRITE|TDF_EXTCOM)
#define ETD_READ (CMD_READ|TDF_EXTCOM)
#define ETD_MOTOR (TD_MOTOR|TDF_EXTCOM)
#define ETD_SEEK (TD_SEEK|TDF_EXTCOM)
#define ETD_FORMAT (TD_FORMAT|TDF_EXTCOM)
#define ETD_UPDATE (CMD_UPDATE|TDF_EXTCOM)
#define ETD_CLEAR (CMD_CLEAR|TDF_EXTCOM)
#define ETD_RAWREAD (TD_RAWREAD|TDF_EXTCOM)
#define ETD_RAWWRITE (TD_RAWWRITE|TDF_EXTCOM)
```

These commands have "TD" in front for Trackdisk device. Here the two commands TD_MOTOR and TD_PROTSTATUS are also defined. They are of interest only to the Trackdisk device.

The CMD_READ command is different. Since bytes are read from every device, it makes sense to standardize the definition of this command for all devices.

The last category of Trackdisk commands are the Extended commands. They have a prefix of "ETD". They differ from the normal commands only through a set bit (TDF_EXTCOM). These extended commands permit some additional features, but require an extended IORequest structure. Because of this, a section has been devoted to these commands.

The Trackdisk device recognizes the following "not extended" commands (the IO section in the IORequest structure contains the individual parameters):

CMD_READ Reads bytes from the disk in the drive.

```
UWORD     io_Command;      CMD_READ
UBYTE     io_Flags;
BYTE      io_Error;        Possible error message.
```

```
ULONG     io_Actual;
ULONG     io_Length;       Number of Bytes to be read.
APTR      io_Data;         Pointer to Data Buffer.
ULONG     io_Offset;       Byte-Offset, where to start
                           read.
```

CMD_WRITE Writes bytes on the disk in the drive.

```
UWORD     io_Command;      CMD_WRITE
UBYTE     io_Flags;
BYTE      io_Error;        Possible error message.
ULONG     io_Actual;
ULONG     io_Length;       Number of Bytes to written.
APTR      io_Data;         Pointer to Data buffer.
ULONG     io_Offset;       Byte-Offset, where write
                           starts.
```

CMD_UPDATE The Trackdisk device always reads entire tracks and stores them in RAM until another track is requested. During a write, data is sometimes only changed in RAM. This command forces an immediate write of the track buffer to the disk if its content has changed.

```
UWORD     io_Command; CMD_UPDATE
UBYTE     io_Flags;
BYTE      io_Error;        Possible error message.
ULONG     io_Actual;
ULONG     io_Length;
APTR      io_Data;
ULONG     io_Offset;
```

CMD_CLEAR The track buffer of the Trackdisk devices is declared invalid so that the track is read again on the next access.

```
WORD      io_Command;      CMD_CLEAR
UBYTE     io_Flags;
BYTE      io_Error;        Possible error message.
ULONG     io_Actual;
ULONG     io_Length;
APTR      io_Data;
ULONG     io_Offset;
```

TD_MOTOR Switches the motor of the device on or off.

```
UWORD     io_Command;      TD_MOTOR
UBYTE     io_Flags;
BYTE      io_Error;        Possible error message.
ULONG     io_Actual;
ULONG     io_Length;       0: Motor off/1:Motor on
APTR      io_Data;
ULONG     io_Offset;
```

TD_FORMAT This command formats one or more indicated tracks. It should be noted that the track offset must be converted to bytes.

```
BYTE     io_Error;        Possible error message.
ULONG    io_Actual;
ULONG    io_Length;       Indicates the number of tracks
                          in Bytes(!).
APTR     io_Data;         Pointer to buffer which contains
                          the track(s).
ULONG    io_Offset;       Points (numbered in Bytes) to
                          the start track.
```

TD_REMOVE An interrupt is initialized every time a disk is removed or inserted.

```
UWORD    io_Command;      TD_REMOVE
UBYTE    io_Flags;
BYTE     io_Error;        Possible error message.
ULONG    io_Actual;
ULONG    io_Length;
APTR     io_Data;         Contains pointer to a software
                          interrupt structure. If a null
                          is passed here, the interrupt is
                          blocked.
ULONG io_Offset;
```

TD_SEEK Moves the read/write heads over the track in which the indicated offset byte is located. No write or read operations occur.

```
UWORD    io_Command;      TD_SEEK
UBYTE    io_Flags;
BYTE     io_Error;        Possible error message.
ULONG    io_Actual;
ULONG    io_Length;
APTR     io_Data;
ULONG    io_Offset;       Byte-Offset for Positioning.
```

TD_CHANGNUM

Returns the counter condition indicating how many times a disk was inserted or removed from the drive.

```
UWORD    io_Command;      TD_CHANGENUM
UBYTE    io_Flags;
BYTE     io_Error;        Possible error message.
ULONG    io_Actual;       Disk-Change-Counter.
ULONG    io_Length;
APTR     io_Data;
ULONG    io_Offset;
```

TD_CHANGESTATE

Tests to see if a disk is in the drive.

```
UWORD    io_Command;      TD_CHANGESTATE
UBYTE    io_Flags;
BYTE     io_Error;        Possible error message.
ULONG    io_Actual;       0: Disk inserted,
                          <>0: Disk removed.
APTR     io_Data;
ULONG    io_Offset;
```

APTR	io_Data;
ULONG	io_Offset;

TD_PROTSTATUS

Tests to see if the inserted disk is write protected.

UWORD	io_Command;	TD_PROTSTATUS
UBYTE	io_Flags;	
BYTE	io_Error;	Possible error message.
ULONG	io_Actual;	0 :Disk not write protected, <>0:Disk is write protected.
ULONG	io_Length;	
APTR	io_Data;	
ULONG	io_Offset;	

TD_RAWREAD

Permits reading of a track without decoding. Can be used to wait for the index hole.

UWORD	io_Command;	TD_RAWREAD
UBYTE	io_Flags;	IOTDF_INDEXSYNC (if waiting for the Index is desired)
BYTE	io_Error;	Possible error message.
ULONG	io_Actual;	
ULONG	io_Length;	Number of data to be read (must be smaller than 32,768).
APTR	io_Data;	Pointer to data buffer
ULONG	io_Offset;	Number of the track to be read (track not cylinder).

TD_RAWWRITE

Permits the writing of a track without decoding. Can be used to wait for the index hole.

UWORD	io_Command;	TD_RAWWRITE
UBYTE	io_Flags;	IOTDF_INDEXSYNC (if wait for Index is desired)
BYTE	io_Error;	Possible error message.
ULONG	io_Actual;	
ULONG	io_Length;	Number of data to be written (must be smaller than 32,768).
APTR	io_Data;	Pointer to data buffer.
ULONG	io_Offset;	Number of the track to be written (track not cylinder).

TD_GETDRIVETYPE

Receives the type of the attached drive.

UWORD	io_Command;	TD_GETDRIVETYPE
UBYTE	io_Flags;	
BYTE	io_Error;	
ULONG	io_Actual;	Type of the drive: DRIVE3_5/DRIVE5_25
ULONG	io_Length;	
APTR	io_Data;	
ULONG	io_Offset;	

TD_GETNUMTRACKS

Receives the number of tracks.

```
UWORD     io_Command;      TD_GETNUMTRACKS
UBYTE     io_Flags;
BYTE      io_Error;
ULONG     io_Actual;       Number of the Tracks of drive.
ULONG     io_Length;
APTR      io_Data;
ULONG     io_Offset;
```

TD_ADDCHANGEINT

A command which appears in principle to be very good. It makes it possible to jump to several interrupts as soon as a disk is removed from the drive or inserted into the drive. The problem with this command is that no Interrupt structure (as in TD_REMOVE) can be passed. The IORequest structure is accepted as an Interrupt structure and is fit into the interrupt list. By inserting a IORequest structure instead of a Interrupt structure, the operating system crashes as soon as a jump occurs to the change interrupt.

TD_REMCHANGEINT

The command is used to remove an Interrupt structure from the Diskchange interrupt list. It is unusable on the basis of an error during the TD_ADDCHANGEINT command because no personal interrupt can be included.

8.3.2 The extended commands

As mentioned, the extended Trackdisk commands offer some extra features. To use them, the IORequest structure must be extended also. It is then known as "IOExtTD" and appears as follows:

```
struct IOExtTD
{                                          /* Offsets   */
struct    IOStdReq  iotd_Req;              /* 0   $00   */
          ULONG     iotd_Count;            /* 48  $30   */
          ULONG     iotd_SecLabel;         /* 52  $32   */
};
```

Basically not much has changed. Only two longwords have been added which the extension can handle.

One is used to confirm that the user has not changed the disk without permission. During normal commands, the Trackdisk device writes a sector without checking if this is the right disk. Since the Trackdisk device counts how many times a disk was inserted or removed, it can test for an unauthorized change.

The program uses TD_CHANGENUM, the current change number, to determine if the right disk is in the drive when reading a sector. If the sector should be changed with ETD_WRITE, the program gives the Changenum through iotd_Count to the Trackdisk device. It now writes the sector to the disk only when the current change number and the one passed agree.

How this is done can be seen in the Bitmap Analysis program which is described in Chapter 6.

The next entry "iotd_SecLabel" is a pointer to a data block which contains, in addition to the actual sector data the sector label data. These are identifications between sectors in a track. Since the Trackdisk device always works on a track basis with the disk, this information is already in the memory. Normally they are not required.

Otherwise all extended commands operate like their "normal" counterparts.

The Include file Devices/Trackdisk.h contains some useful constants for working with the disk:

```
#define NUMCYLS      80                   Number of cylinders.
#define NUMSECS      11L                  Sectors per track.
#define NUMHEADS     2                    Heads per drive.
#define MAXRETRY     10                   Max. repetitions on error.
#define NUMTRACKS (NUMCYLS*NUMHEADS)      Tracks per disk.
#define NUMUNITS     4L                   Devices connected.
#define TD_SECSHIFT 9L                    Size of label area.
#define TD_NAME "trackdisk.device"        Name of device.
```

8.4 The Trackdisk structures

To be able to understand the routines for disk control, as they are used by the operating system the most important structures are presented here. They are the Trackdisk Device structure and the Message Port structure.

The Device structure

Within the Device structure is data which is required for the organization of all the disk operations. This includes the pointers to additional structures (Msg. Port structures), which control the attached drives and take over the work.

The Msg. port structure

There is a Msg. Port structure for every attached drive. Besides the "naked" Port structure which is familiar from the C Include files, there are additional interesting entries which must be explained for the following chapters. These additional entries are important for the control of the drive through the operating system.

More details are provided in the following sections in the structure entries.

8.4.1 The Device structure

The first explanation concerns the construction of the Trackdisk Device structure. We'll discuss some, but not all of the entries for this structure.

Offset		Entry	Explanation
00	$00	struct Library	Library structure (valid for all Device structures).
34	$22	$0000	Word to access longword-address.
36	$26	*Msg-Port 0	Pointer to Msg.-Port for Drive 0.
40	$28	*Msg-Port 1	Pointer to Msg.-Port for Drive 1.
44	$2E	*Msg-Port 2	Pointer to Msg.-Port for Drive 2.
48	$30	*Msg-Port 3	Pointer to Msg.-Port for Drive 3.
52	$34	*ExecBase	Pointer to ExecBase.
56	$38	*GfxBase	Pointer to GfxBase.
60	$3C	*DSKResource	Pointer to Disk-Resource.
.			
.			
78	$4E	*Timer.device	Pointer to Timer-Device.
.			
.			
94	$5E	*ciab.resource	Pointer to CIAB-Resource.
.			
.			

If a drive is not attached, the pointer to the corresponding Msg. port is null.

8.4.2 The Port structure

Next is the description of those parts of the Msg. Port structure which are important.

Offset		Entry	Explanation
00	$00	struct MsgPort	Usual Msg. Port structure.
34	$22	unit_Flags	Flagbits used for the control of the device. Bit 0 = 1 => Device busy
35	$23	unit_Pad	Empty byte to reach even addresses.
36	$24	unit_OpenCnt	Counter for number of the tasks which access the device.
38	$26	Change1	First compression value for writing a track if the track number is higher than the value indicated here (80).
40	$28	Change2	Second compression value for track indicated (not used ($FFFF)).
42	$2A	Change3	Third compression value for track indicated (not used ($FFFF)).
44	$2C		StepTimeValue for time loop during stepping of head (3000).
48	$30	Wait	Value for time loop after the desired track is reached (6000).
52	$34	ErrorNum	Number of errors permitted during disk access (10)
	$35	Drive_Type	Type of the attached drive (1 for 3.5 disk):
	$36	TrackNum	Number of accessible tracks (160)
56	$38	MaxOffset	Largest offset for disk = $DC000 => 160 Tracks
	.		
	.		
64	$40	Flagbits	Control Bits: Bit 1 = 1 => Drive empty Bit 2 = 1 => Extended command Bit 3 = 1 => Close Device Bit 4 = 1 => Disk is protected
65	$41	DriveBit	Drivebits for Drive select register related to motor.

```
66   $42 ErrorCNT          Counter for number of errors
                           during disk access.
67   $43 DriveNum          Drive number.
68   $44 *IoRequest        Pointer to IoRequest structure
                           passed.
72   $48 Sector            Sector number to be read or
                           written.
74   $4A Track             Track number of the head.
76   $4C Track             Track number of the head.
78   $4E *LoadBuffer       Pointer to buffer into which
                           the data from the disk is
                           written (MFM-coded data).
82   $52 *SaveBuffer       Same as LoadBuffer, but write
                           buffer.
90   $5A *Headbuffer       Pointer to data buffer, when
                           the 16 empty bytes in the
                           Block-Header should be decoded.
                           Otherwise not set.
94   $5E struct IoRequest  IoRequest structure which is
                           sent to the timer device during
                           disk operations.
134 $86 struct IoRequest   IoRequest structure which is
                           sent to the disk in the drive
                           during maintenance.
174 $AE struct MsgPort     Port to which messages are sent
                           when a certain process is
                           finished, for example disk
                           block ready.
208 $D0 struct Message     Message sent to the port
                           (Offset 174) to indicate the
                           end of a process.
228 $E4 struct interrupt   Structure for Disk-Block-Int.
     242     $F2           is_Data is_Data for Disk-Block-
                           Int.
     246     $F6           is_Code is_Code for Disk-Block-
                           Int. ($FEA6F2)
250 $FA struct Interrupt   structure for DiskSYNC-Int.
     264     $108          is_Data is_Code for DiskSYNC-
                           Int.
     268     $10C          is_Code is_Code for DiskSYNC-
                           Int.
272 $110struct Interrupt   Structure for Index-Interrupt
        286  $11E          is_Data is_Data for Index-Int.
        290  $122          is_Code is_Code for Index-Int.
                           ($FEB38E)
298 $12A  CangeCNT         Incremented when the disk is
                           removed from the drive or
                           inserted in it.
```

With the help of these two structures, the routines which are used by the operating system, related to the drive, can be analyzed.

In the following text, the Msg. port is also called Drive port.

8.4.3 The Resource structure

The third structure is the Trackdisk Resource structure which is used to control the drive in connection with multitasking.

Some functions are available for work on the structure which are accessed like a library through jumps with negative offsets. Jumps are made to all functions with the base address of the Resource structure in A6.

The functions have the following significance:

Offset -6 $FC4A62
Function: Sets bit for drive.

Parameter passed:
D0 = Drive number.

Return parameter:
D0 = $FF Drive was not yet present.
D0 = $00 Drive was present.

Offset -12 $FC4A6E
Function: Erases bit for drive.

Parameter passed:
D0 = Drive number

Offset -18 $FC4996
Function: Announce drive.

If a drive is accessed by the operating system, access is prevented from another task to this or another drive at the same time since the access would occur through the same hardware registers. This would disturb access to the drive by the first task. To prevent this interference, a wait must occur for the release of the registers. The request to use the registers is attached to the end of a list. If the list is not empty, the task goes into a waiting position until the registers are released.

The message to wait for the release of the registers is the Message structure from the Drive Port structure with the offset 208:

Parameter passed
A1 = Pointer to message to be added.

Parameter returned:
D0 not equal to zero means that no wait is required. Access can start immediately.

Offset -24 $FC4A0E
Function: Remove drive.

One Trackdisk task releases the hardware registers for the others and sends a message to the next one on the list so it can continue its work.

Offset -30 $FC4A74
Function: Test if drive is present.

Parameter passed:
D0 = Drive number

Parameter returned:
D0 = $0000 Drive present.
D0 = $FFFF Drive not present.

The data area of the structure appears as follows:

```
00 $00 struct Library       Standard Library structure as
                            in all Resource structures.
34 $22 *Reply-Message       If a drive has been
                            registered, this is the
                            pointer to the Message
                            structure (Offset 208) in the
                            Port structure.
38 $26 DriveBits            A set Bit signals a drive
                            present. If Bit 7 is set, a
                            drive is registered.
39 $27
40 $28 *ExecBase            Pointer to ExecBase structure.
44 $2C *ciab.resource       Pointer to CIAB resource.
48 $30 60 $3C               Words which signal if the
                            drive is connected.
                            $FFFF => no Drive
                            $0000 =>  Drive
64 $40 struct List          List for drive registration.
```

8.5 The internal processing of command parameters

Now that we have described the programming for the Trackdisk devices and structures, the internal processing of the commands which are sent through the IORequest structure will now be covered.

It's assumed that the IORequest structure has been initialized and the device was opened.

8.5.1 The DoIO function

In A1 is a pointer to the previously created IORequest structure.

```
fc06dc move.l  A1,-(A7)        Save A1
fc06de move.b  #$01,30(A1)     Set Quick-Bit
fc06e4 move.l  A6,-(A7)        Save A6
fc06e6 move.l  20(A1),A6       Get Pointer to Device
```

The following jump into the routine which sends the command (the IORequest structure) to the device, is discussed later.

```
fc06ea jsr     -30(A6)         Jump to IO execution
fc06ee move.l  (A7)+,A6        Get A6
fc06f0 move.l  (A7)+,A1        Get A1
```

The WaitIO function which is used by the DoIO function starts here.

```
fc06f2 btst    #0,30(A1)           Test Quick-Bit
fc06f8 bne.s   $fc0744             Done when set
fc06fa move.l  A2,-(A7)            Save A2
fc06fc move.l  A1,A2               Pointer to IORequest to A2
fc06fe move.l  14(A2),A0           Pointer to Reply-Port
fc0702 move.b  15(A0),D1           Get signal bit for Port
fc0706 moveq   #$00,D0             Erase D0
fc0708 bset    D1,D0               Set Bit for Signal
fc070a move.w  #$4000,$dff09a      Disable-
fc0712 addq.b  #1,294(A6)          Macro
fc0716 cmpi.b  #$07,8(A2)          Type of Msg. = ReplyMsg.?
fc071c beq.s   $fc0724             Branch if Type ok
fc071e jsr     -318(A6)            else wait for Msg. ( Wait() )
fc0722 bra.s   $fc0716             indeterminate Jump
fc0724 move.l  A2,A1               IORequest to A1
fc0726 move.l  (A1),A0
fc0728 move.l  4(A1),A1            remove Node from Reply-Msg-List
fc072c move.l  A0,(A1)
```

```
fc072e move.l  A1,4(A0)
fc0732 subq.b  #1,294(A6)         Enable-
fc0736 bge.s   $fc0740
fc0738 move.w  #$c000,$dff09a     Macro
fc0740 move.l  A2,A1              Pointer to IORequest to A1
fc0742 move.l  (A7)+,A2           Create A2
fc0744 move.b  31(A1),D0          Error-Flag to D0
fc0748 ext.w   D0                 Extend Sign
fc074a ext.l   D0                 Extend Sign
fc074c rts                        Return Jump
```

In the routine above, the quick bit is set first. Then the pointer to the device is put in A6 and a jump is performed to the BeginIO function.

This terminates the passing of the command. All that remains is to wait for completion. If the quick bit was not reset, the routine is now finished. Otherwise a test is made if the IO process was completed. For this test it is sufficient to test the type of Message structure for "Reply Msg." If this is not the case, the task goes to a wait condition until a proper message arrives.

8.5.2 The BeginIO function

In this function a command to be executed is tested for validity and passed to the Trackdisk task. The routine also tests if the command sent can be executed directly, or if it must be sent to the device.

When the program returns from this routine, the message type in the IORequest structure is always set to "Message". The IORequest structure is always passed to the Trackdisk task as a message in the routine. Passing occurs through the Message Port structure belonging to the drive and therefore the task.

The function jumps from the DoIO function with "JSR -30(A6)".

The pointer to the IORequest structure is in A1.

The pointer to the device is in A6.

```
fe9fbe clr.b   31(A1)          Erase Errorflag
fe9fc2 moveq   #$00,D0         Clear D0
fe9fc4 move.b  29(A1),D0       io_Command to D0
fe9fc8 cmpi.b  #$16,D0         Command permitted ?
fe9fcc bcc.s   $fea016         branch if illegal (command > 21)
fe9fce move.l  24(A1),A0       Pointer to Device-Port
fe9fd2 move.l  #$000c61c2,D1   Command decode bits
fe9fd8 btst    D0,D1           Execute command directly ?
fe9fda bne.s   $fe9ff0         yes, execute command
fe9fdc andi.b  #$7e,30(A1)     Erase flags up to Quick-Bit
fe9fe2 move.l  A6,-(A7)        save A6
fe9fe4 move.l  52(A6),A6       get ExecBase
```

```
fe9fe8 jsr      -366(A6)        PutMsg (Pass IORequest to
                                Trackdisk-task)
fe9fec move.l   (A7)+,A6        get A6
fe9fee bra.s    $fea014         unconditional Jump
fe9ff0 bset     #7,30(A1)       Set flag for execution
fe9ff6 move.b   #$05,8(A1)      Type in IORequest structure
                                to "Message",
                                to make WaitIO wait
fe9ffc movem.l A3-A2,-(A7)      save A2 and A3
fea000 move.l   A0,A3           Pointer to Drive-Port to A3
fea002 move.l   A1,A2           Pointer to IORequest to A2
fea004 lea      762(PC)(=$fea300),A0 pointer to command tab.
fea008 lsl.w    #2,D0           command *4, to get Offset
fea00a move.l   0(A0,D0.W),get A0 jump
fea00e jsr      (A0)            Jump
fea010 movem.l (A7)+,A3-A2      restore A2 and A3
fea014 rts                      Return jump

fea016 bsr.l    $fea06e         Error Output
fea01a bar      $fea014         Return jump
```

As already mentioned the routine tests if the command can be executed. Then follows a listing of commands which cannot be passed to the device.

These commands are again divided into two sub-groups: the CMD and the TD commands. The CMD commands which follow aren't permitted for the Trackdisk device and are terminated during the direct call immediately with the error number 253 ($FD).

```
CMD_RESET
CMD_STOP
CMD_START
CMD_FLUSH
```

In contrast with the CMD commands which are not permitted, are the allowable TD commands. The following are executed directly.

```
TD_CHANGENUM
TD_CHANGESTATE
TD_GETDRIVETYPE
TD_GETNUMTRACKS
```

8.5.3 The Trackdisk task

As described, most commands are passed from the BeginIO function to the Trackdisk task. This transmission occurs with the help of the message port which is "coupled" to the Trackdisk task.

The main task routine is busier than you would suspect because it has other duties besides processing the commands coming from the BeginIO function. Besides the IORequest structures, the task also receives additional messages.

This message ensures that every half-second the task tests to see if a disk was removed from its assigned drive. If this has happened, the heads are moved to cylinder zero and the delay with which the message wakes the task from its rest position is set for 2.5 seconds. The message, which now arrives every 2.5 seconds, causes the task to check if a disk was inserted in the meantime. The Amiga is (minimally) faster if there is no disk in the drive.

The task differentiates between the two messages and passes them on, with the following exception: If the message was sent by the CloseDevice() function, this task is not passed on, but executed directly in the main routine.

To permit each of the four possible tasks the opportunity of addressing a disk drive, only one program is needed in memory; in this case in the Kick-ROM. Every task accesses the same program. This program is documented as follows:

```
FEAE50 MOVE.L  8(A7),A6        Pointer to Track device
FEAE54 MOVE.L  4(A7),A3        Pointer to Track port
FEAE58 LEA     302(A3),A0      Pointer to Track task
FEAE5C MOVE.L  A0,16(A3)       Enter Task as Msg.task for Port
FEAE60 BSR.L   $FE9960         check if disk was removed
```

Start of the loop in which the task runs until a message is sent.

```
FEAE64 BSR.S   $FEAE7A         test for Msg. and process
FEAE66 MOVE.L  #$00000300,D0   Bits, for which the task is waiting
FEAE6C MOVE.L  A6,-(A7)        Save pointer to device
FEAE6E MOVE.L  52(A6),A6       Get ExecBase
FEAE72 JSR     -318(A6)        Function: Wait()
FEAE76 MOVE.L  (A7)+,A6        Restore pointer to device
FEAE78 BRA.S   $FEAE64         unconditional jump
```

In the following program a message portion is obtained from port and processed.

A3 = Pointer to Msg. Port for the Drive.
A6 = Pointer to the Trackdisk Device structure.

```
FEAE7A BSET    #0,34(A3)       Set Flag for Task
FEAE80 BNE.L   $FEAF4A         End if Task is already working
FEAE84 MOVE.L  A3,A0           Pointer to Port to A0
FEAE86 MOVE.L  A6,-(A7)        save A6
FEAE88 MOVE.L  52(A6),A6       get ExecBase
FEAE8C JSR     -372(A6)        Function: GetMsg()
FEAE90 MOVE.L  (A7)+,A6        restore A6
FEAE92 TST.L   D0              Message present ?
FEAE94 BEQ.L   $FEAF3E         branch if no Msg.
FEAE98 MOVE.L  D0,A2           Pointer to Message to A2
FEAE9A BCLR    #3,64(A3)       Flag for Close-Device
```

```
FEAEA0 BEQ.L    $FEAF1E          branch if no Close
FEAEA4 MOVE.L   82(A3),A0        Storage buffer to A0
FEAEA8 BCLR     #0,2(A0)         Was Buffer changed ?
FEAEAE BEQ.L    $FEAEBA          branch if not changed
FEAEB2 MOVE.L   A0,78(A3)        Storage buffer = Load buffer
FEAEB6 BSR.L    $FEA958          write Track
FEAEBA MOVE.L   82(A3),A0        Write buffer to A0
FEAEBE MOVEQ    #$FF,D0          -1 to D0
FEAEC0 MOVE.W   D0,0(A0)         mark Track as invalid
FEAEC4 MOVE.W   D0,76(A3)        mark Track as invalid
FEAEC8 MOVEQ    #$00,D0          Value for motor off
FEAECA BSR.L    $FEA462          Motor off
FEAECE MOVE.L   A6,A0            save A6
FEAED0 MOVE.L   52(A0),A6        get ExecBase
FEAED4 ADDQ.B   #1,295(A6)       Forbit
FEAED8 MOVE.L   A0,A6            restore A6
FEAEDA TST.W    36(A3)           remove Drive ?
FEAEDE BNE.L    $FEAF12          branch if not removed
```

The drive is removed:

```
FEAEE2 MOVEQ    #$00,D0          clear D0
FEAEE4 MOVE.B   67(A3),D0        Drive number to D0
FEAEE8 MOVE.L   A6,-(A7)         save A6
FEAEEA MOVE.L   60(A6),A6        Pointer to Disk-Resource
FEAEEE JSR      -12(A6)          erase Motor-Bit
FEAEF2 MOVE.L   (A7)+,A6         restore A6
FEAEF4 LEA      36(A6),A0        Pointer to Driveports
FEAEF8 MOVEQ    #$00,D0          erase D0
FEAEFA MOVE.B   67(A3),D0        get Drive number
FEAEFE LSL.L    #2,D0            determine Position of
FEAF00 ADDA.L   D0,A0            the Pointer to Port
FEAF02 CLR.L    (A0)             erase Pointer
FEAF04 SUBA.L   A1,A1            clear A1
FEAF06 MOVE.L   A6,-(A7)         save A6
FEAF08 MOVE.L   52(A6),A6        get ExecBase
FEAF0C JSR      -288(A6)         Function: RemTask() (remove
                                 own Task)
FEAF10 MOVE.L   (A7)+,A6         restore A6

FEAF12 MOVE.L   A6,-(A7)         save A5
FEAF14 MOVE.L   52(A6),A6        get ExecBase
FEAF18 JSR      -138(A6)         Function: Permit()
FEAF1C MOVE.L   (A7)+,A6         restore A6
```

Check where message originated:

```
FEAF1E MOVE.L   A2,A1            get Pointer to Message
FEAF20 LEA      134(A3),A0       Pointer to Message from Timer
FEAF24 CMPA.L   A0,A2            is Msg. from Timer
FEAF26 BNE.S    $FEAF30          no, then command message
FEAF28 BSR.L    $FE9960          test if Disk was removed
FEAF2C BRA.L    $FEAE84          get new Message
```

The part of the program which processes an IO structure sent by the programmer begins here:

```
FEAF30 BSET     #1,34(A3)        Set Bit for processing of a command
FEAF36 BSR.L    $FEA01C          process command
FEAF3A BRA.L    $FEAE84          get new Message
```

```
FEAF3E BCLR    #1,34(A3)       clear Flags for
FEAF44 BCLR    #0,34(A3)       processing of commands
FEAF4A RTS                     Return Jump
```

8.5.4 Differentiating the commands

On the basis of the routine just described it can be seen that for a message sent by the timer, a branch is taken to $FE9960. In contrast the routine is continued at $FEA01C, when a command arrives. Of interest is the portion starting at $FEA01C, which controls the IO structure.

Pointer in A0 to the IO Request structure.
Pointer in A3 to the Drives port.
Pointer in A6 to the Device structure.

```
FEA01C MOVE.L  A2,-(A7)          save A2
FEA01E MOVE.L  A1,A2             IO-Request to A2
FEA020 ANDI.B  #$FA,64(A3)       erase Status-Bits
FEA026 BSR.L   $FE998C           test for Disk in Drive
FEA02A MOVE.L  A2,A1             IO-Request-Structure to A1
FEA02C MOVE.W  28(A2),D0         io_Command to D0 (command)
FEA030 BTST    #15,D0            extended command ?
FEA034 BEQ.S   $FEA052           branch if not extended
FEA036 BSET    #2,64(A3)         set Bit for extended command
FEA03C MOVE.L  294(A3),D1        number of Disk changes to D1
FEA040 CMP.L   48(A2),D1         compare with iotd_Count
                                 (within IOExtTD-Structure)
FEA044 BLS.S   $FEA052           branch if value still OK
FEA046 MOVE.B  #$1D,31(A2)       otherwise Disk changed too often
FEA04C BSR.L   $FEA1B0           pass errors
FEA050 BRA.S   $FEA066           unconditional Jump

FEA052 MOVEQ   #$00,D1           clear D1
FEA054 MOVE.B  D0,D1             command to D1
FEA056 LSL.W   #2,D1             determine Offset for command
table
FEA058 LEA     678(PC)(=$FEA300),A0  Pointer to Table
FEA05C MOVE.L  0(A0,D1.W),A0      get Address for command
FEA060 JSR     (A0)               call command
FEA062 BSR.L   $FE998C            test for Disk in Drive
FEA066 MOVE.L  (A7)+,A2           restore A2
FEA068 RTS                        Return Jump
```

The start addresses of the commands which can be called by the Trackdisk device, are in a table starting at $FEA300.

The following listing shows the jump locations for various functions:

```
$FEA06E => CMD_INVALID     no
$FEA06E => CMD_RESET       no Function (Error).
$FEA734 => CMD_READ
$FEA734 => CMD_WRITE
$FEAAAA => CMD_UPDATE
$FEAA94 => CMD_CLEAR
$FEA06E => CMD_STOP        no Function (Error).
$FEA06E => CMD_START       no Function (Error).
$FEA06E => CMD_FLUSH       no Function (Error).
$FEA9FE => TD_MOTOR
$FEAA14 => TD_SEEK
$FEA07A => TD_FORMAT
$FE9AB6 => TD_REMOVE
$FE9A96 => TD_CHANGNUM     Executed directly.
$FE9AA2 => TD_CHANGESTATE  Executed directly.
$FEAA44 => TD_PROTSTATUS
$FEB2E8 => TD_RAWREAD
$FEB2EE => TD_RAWWRITE
$FEB3B6 => TD_GETDRIVETYPE        Executed directly.
$FEB3C8 => TD_GETNUMTRAKS  Executed directly.
$FE9AC2 => TD_ADDCHANGEINT
$FE9ADE => TD_REMCHANGEINT
```

8.6 The RAW commands
(with Index interrupt)

Some of the information introduced next will be comprehended fully only after reading Chapter 9 (Direct disk access). It has been inserted here only because it fits more logically into the outline.

As will be shown in Chapter 9, it isn't possible for the operating system to load data from the disk with the controller synchronized. The operating system normally loads without synchronization. The system can be forced into synchronization in connection with the RAW command.

To understand how this can be done the documented RAW functions are shown:

Jump to TD_RAWREAD:

```
feb2e8 lea      -3522(PC)(=$fea528),A0  Pointer to Routine
                                        for reading Track
feb2ec bra.s    $feb2f2                 unconditional Jump
```

Jump to TD_RAWWRITE:

```
feb2ee lea      -3384(PC)(=$fea5b8),A0  Pointer to Routine
                                        for writing Track
feb2f2 movem.l  A4-A2/D3-D2,-(A7)  save Register
feb2f6 link     A5,#-8             make space in Stack
feb2fa move.l   A1,A2              IORequest to A2
feb2fc move.l   A0,A4              Read/Write routines to A4
feb2fe moveq    #$01,D0            Value for Motor on
feb300 bsr.l    $fea462            switch motor on
feb304 move.l   44(A2),D0          get Track-Number from IORequest
feb308 cmp.l    54(A3),D0          Number legal ?
feb30c bcc.s    $feb35c            branch if Track too high
feb30e bsr.l    $fea3da            position head on Track
feb312 move.b   D0,31(A2)          enter Error in Error-Flag
feb316 bne.s    $feb34e            branch if Error
feb318 move.l   36(A2),D0          number of Bytes to be
                                   read/written
feb31c cmp.l    $008000,D0         more or equal to $8000 Bytes
feb322 bcc.s    $feb35c            branch if larger
feb324 move.l   40(A2),A0          Pointer to Data buffer
feb328 lea      -3456(PC)(=$fea5aa),A1  Pointer to Routine for
                                   switching on DMA
feb32c btst     #4,30(A2)          IOTDF_INDEXSYNC in Flags
                                   set ?
feb332 beq.s    $feb348            branch if not set
feb334 lea      -8(A5),A1          Pointer to place in Stack
feb338 move.l   A1,D1              Pointer to D1
feb33a lea      82(PC)(=$feb38e),A1   Pointer to Interrupt
                                     Routine by Index-Sync
```

```
feb33e movem.l A1/D1,286(A3)           enter values for is_Data and
                                       is_Code
                                       for Index-Interrupt
feb344 lea     30(PC)(=$feb364),A1     Pointer to Start-
                                       Routine for DMA
feb348 jsr     (A4)                    jump to read/write routine
feb34a move.b  D0,31(A2)               write Error in Error-Flag
feb34e move.l  A2,A1                   Pointer to IORequest to A1
feb350 bsr.l   $fea1b0                 answer IORequest
feb354 unlk    A5                      release Stack again
feb356 movem.l (A7)+,A4-A2/D3-D2       Restore Registers
feb35a rts                             Return Jump

feb35c move.b  #$fc,31(A2)             Move Error into Error-Flag
feb362 bra.s   $feb34e                 unconditional Jump
```

Next follows the routine to which a jump occurs if read/write is started through the DMA, if index synchronization should be on.

```
feb364 move.l  286(A3),A0      get is_Data (pointer to stack)
feb368 movem.l A6/D0,(A0)      Number of bytes to be read and
                               enter pointer to device
feb36c moveq   #$10,D0         value for Index-Int.
feb36e move.l  A6,-(A7)        save A6
feb370 move.l  94(A6),A6       pointer to CIAB resource
feb374 jsr     -24(A6)         If interrupt present
                               clear interrupt
feb378 move.l  (A7)+,A6        restore A6
feb37a move.l  #$00000090,D0   permit value for Index int.
feb380 move.l  A6,-(A7)        save A6
feb382 move.l  94(A6),A6       Pointer to CIAB-Resource
feb386 jsr     -18(A6)         permit Interrupt
feb38a move.l  (A7)+,A6        restore A6
feb38c rts                     Return Jump
```

After the index interrupt (CIAB flag) is released, the task waits, which can be seen in the documented Load routine in Chapter 9.

If the index marking sets the flag line of the CIAB, an interrupt is triggered which is handled by the CIAB Resource structure. For this purpose the following entries are in the CIAB Resource structure starting at Offset 112 ($70):

CIAB resource

Offset	Significance
112 $70	Pointer to disk resource
116 $74	Pointer to the program to be executed
120 $78	Pointer to the Interrupt structure (not important)

Starting at offset 116 there is the pointer to the program to be executed as soon as an interrupt is pending. The program starts at $FC4AB0.

A1 is the pointer to disk resource.

```
fc4ab0 move.l  34(A1),D0       Was Drive registered ?
fc4ab4 beq.s   $fc4ad8         End, if not registered
fc4ab6 move.l  D0,A1           Reply-Msg. to A1
fc4ab8 movem.l 78(A1),A5/A1    get Pointer to Data buffer and
jump for
```

```
fc4abe jmp      (A5)              Jump to Interrupt,
                                  normally $FEB38E
   .
   .
fc4ad8 rts
```

With offset 78 in the Reply Msg. (which starts at offset 208 in the Drives Port structure) therefore in $FC4AB8, a pointer is obtained to the is_Data entry (Offset 208+78 = 286 = $11E in drives port), which was set by the previously documented routine in $FEB364. The following longword is "is_Code" which represents the pointer to the Interrupt program. This program starts the reading of the track.

The actual Interrupt program starts at $FEB38E and has the following appearance:

A1 is the pointer to the buffer which is in the stack (see $FEB2F6, $FEB334 and following).

```
feb38e move.l  A2,-(A7)         save A2
feb390 move.l  A1,A2            Pointer to Buffer to A2
feb392 move.l  (A2),D0          number of data to be read
                                (see $FEB368)
feb394 lea     $dff000,A1       Pointer to Custom-Chips
feb39a bsr.l   $fea5aa          start DMA
feb39e move.l  4(A2),A0         Pointer to Track-Device
                                (see $FEB368)
feb3a2 move.l  94(A0),A0        Pointer to CIAB-Resource
feb3a6 moveq   #$10,D0          block value for Index Int.
feb3a8 move.l  A6,-(A7)         save A6
feb3aa move.l  A0,A6            CIAB rsource to A6
feb3ac jsr     -18(A6)          block Index interrupt
feb3b0 move.l  (A7)+,A6         restore A6
feb3b2 move.l  (A7)+,A2         restore A2
feb3b4 rts                      Return Jump
```

Admittedly the entire process is somewhat confusing, but permits waiting for the index mark to synchronize the data read.

The synchronization of the data is of value when it concerns reading copy protected data, reading a foreign format, etc. Using the operating system routines is simpler than writing completely new routines.

To switch on synchronization of the byte to be read from disk, the previously described Index interrupt must be redirected to the user routine. The synchronization, which is always switched off by the operating system, can be retroactively switched on.

The best chance to redirect the interrupt to a proprietary routine, is at a location where it jumps through the vector in the CIAB Resource structure (offset 116 = $74) to the routine at $FC4AB0.

From there a branch occurs to the last Interrupt routine which performs the starting of DMA. Since the interrupt is now in the user's routine, synchronization can be switched on before the DMA is switched on. A decision can also be made to read GCR format instead of MFM format without having to write a long user routine.

The redirection of the interrupt and switching on of the synchronization occurs as follows:

```
FindName        = -276
ResourceList    =  336
is_Code         =  116

        move.l $4,a6             ;get ExecBase
        lea ResourceList(a6),a0  ;Pointer to Resource-
                                 ;List in ExecBase
        lea ResName,a1           ;Pointer to Name
        jsr FindName(a6)         ;search for Resource
        tst.l d0                 ;found ?
        beq Error                ;no, Error
        move.l d0,a1             ;ResourceBase to A1
        lea Program,a0           ;Pointer to Interrupt
                                 ;Program
        move.l a0,is_Code(a1)    ;enter Pointer to Resource
Error:      rts                  ;Return Jump

ResName:     dc.b "ciab.resource",0    ;Resource-Name
            align.w              ;bring following Commands
                                 ;to even Addresses

;The Interrupt program, which is called by the
;routines of the CIAB resource and performs the
;Synchronization, starts here.

Program:    move.l 34(a1),d0        ;Drive registered
            beq pr1                 ;no, End
            move.l d0,a1            ;Reply-Msg. to A1
            movem.l 78(a1),a5/a1    ;get Jump for Interrupt
            move.w #$8400,$dff09e ;switch on Word sync
            move.w #$4489,$dff07e ;pass Sync-Word
            jmp (a5)                ;start DMA
pr1:        rts                     ;Return Jump

                END
```

9.
Accessing the disk without DOS

9. Accessing the disk without DOS

Now that you've learned how to access the disk with the help of DOS, direct mode access independent of DOS will be discussed. Following this, the formats used by the Amiga will be discussed. Before starting on these matters, an explanation of how the data is stored on the disk must be provided so the sections which follow can be understood.

9.1 The recording format on the disk

To understand how information is stored on disk, it is necessary to become familiar with the basic principle of storing data on magnetic surfaces (cassette, tape, hard disk, disk).

First of all you have to know that a magnetic field is created by a current passing through a coil. This is why a disk should not be stored near a transformer, speaker or electric motor.

Induction

Current passing through a coil produces a magnetic field. On the other hand a magnetic field acting on a coil produces a current. This phenomenon is called *induction.*

A recording head, such as the read/write head of the disk drive, is basically nothing more than a coil through which current passes to produce a magnetic field. The condition of the magnetic field, on or off, is stored on disk.

Data is dissected into its smallest units (bits) for recording on a disk. These bits can assume only two states, 0 or 1. The condition 1 is represented by the presence of current, and the condition 0 by the absence of current.

The data is sent to the r/w head (read/write head) of the drive in the form of current impulses with a timer providing the intervals in which an impulse is sent. These impulses are recorded on the surface of the disk in the form of magnetization (north or south pole).

When the read head rides over the disk, the magnetized coating on the surface of the disk creates impulses which are restored into bytes by the electronics of the drive.

The data is stored on a disk in a form different from what you would expect. The 0 and 1 bits are represented not only by a different magnetization (for example north pole = 1 and south pole = 0), but also their change. A change in magnetization represents a 1 bit and unchanging magnetization within a certain time interval represents a 0 bit.

9.2 The MFM and GCR formats

As mentioned, a 1 bit is represented by a changing magnetization and a 0 bit through a steady magnetization. The writing of data with this method creates a problem. The phases of the unchanging magnetization cannot be too long or the controller goes out of synchronization. This is because of variations in the drive since its "orientation" on the disk. How is it possible to store data which consist partly of nulls?

To do this, the data to be written must be coded before being recorded. The coding must be in such a manner that not too many null bits are recorded consecutively.

Sync marking

There is another reason for coding data which must be written on a disk. To read data from a disk, the controller must know where to start with the reading, i.e. where the data starts. To mark the start of data a bit combination is needed which cannot occur in normal data. A combination of this type is called *synchronization marking* or *sync marking*.

The required sync marking is also the second reason why data must be coded because any combination of data can occur. There are two different systems for coding data on the Amiga.

9.2.1 The MFM format

The Amiga uses MFM coding for the encryption of data. For coding of data according to this system, data is recorded on the disk in the form of data bits. In addition clock bits are recorded to insure that the controller does not get out of synchronization.

With this system every data bit follows a clock bit, doubling the number of bits which must be written. This is not a space-efficient system.

The system for setting the clock bits is relatively simple. If one of the adjacent data bits is set, a reset clock bit is inserted. If the neighboring data bits are reset, a set clock bit is inserted.

For the coding of the byte $A1, the coded word has the following appearance.

```
Byte      Bitpattern
$A1       %10100001
```

The clock bits appear as follows:

```
Data Bits    % 1 0 1 0 0 0 0 1
Clock Bits   %0 0 0 0 1 1 1 0
Result       %0100010010101001  = $44A9
```

This coding system prevents too many null bits following each other. It also prevents two or more 1 bits from following each other. This is important since the controller is not capable of recognizing a change in magnetization which occurs too quickly, without errors. This is also the reason why the GCR format, described at the end of this chapter, can only write with half the speed.

After the discussion of the formats, the synchronization of the controller will be described. For synchronization the disk is searched for a word which the user has provided. This word cannot be present in normal data since the controller would synchronize at the wrong place. Data must be found which cannot be reached with normal coding and which can be recognized by the controller without problems. Such a combination is a sequence of three bits set to null, where two of them are data bits (normally between reset data bits there is always a set clock bit). For example the combination $4489 is used by DOS as a marker.

```
                    D D
$4489  =  %0100010010001001
                     T
```

The word is illegal. It can never occur through normal coding.

After the controller has found this word, it knows that data starts here and reads it without error. It does not matter if these are sync words or legal data. A renewed synchronization is possible only after the completion of the read process.

9.2.2 The GCR format

The second format which the controller must process is the GCR format (Group Code Recording), which is not used by DOS. It occupies significantly less space on the disk, but has a disadvantage.

In the GCR format groups of four bits are coded into a combination of five bits. This eliminates too many null bits following each other. After the coding there are never more than two null bits or more than eight one bits following each other.

The problem is that several one bits can follow each other in this format and the controller cannot process these. This has the result of changing the data recording density. To work with the GCR format without error, a switch is made to half the recording speed (from 2 ms to 4 ms).

The following table shows the coding according to this system:

Hexadecimal	Binary	GCR equivalent:
$0 (0)	0000	01010
$1 (1)	0001	01011
$2 (2)	0010	10010
$3 (3)	0011	10011
$4 (4)	0100	01110
$5 (5)	0101	01111
$6 (6)	0110	10110
$7 (7)	0111	10111
$8 (8)	1000	01001
$9 (9)	1001	11001
$A (10)	1010	11010
$B (11)	1011	11011
$C (12)	1100	01101
$D (13)	1101	11101
$E (14)	1110	11110
$F (15)	1111	10101

Byte $39 is coded according to this system as follows:

```
$39  =  %0011 1001  <=>  10011 11001  <=>  1001 1110 01
          $3   $9                           $8   $E  --
```

Two bits remain as "excess" since they cannot be gathered into a byte. Blocks of four bytes are coded to five bytes to avoid this problem.

Now that the coding system is understood, how about synchronization? It's impossible in this system to have more than eight 1 bits following each other. Such a combination cannot be created through coding of data. This is used by the controller, which recognizes the appearance of nine or more 1 bits sequentially as a synchronization marker.

Reading of data is suspended until a null bit is found after the recognition of the sync marker.

When data is written in this format, it is important to note that the data beginning after the sync marker always starts with a null bit. If this is not the case, the first data bits are recognized as being part sync marker and the following data is shifted by the corresponding number of bits.

The writing of data according to this system has the following appearance.

```
$FFFF.........FF55.........Data

Sync         Sync-End        Data
```

9.3 Construction of a track

A track consists of 11 blocks of 512K each and two are used as pointers to the next one.

The data is stored in MFM format. In addition to the data there is other information on the track which is used by DOS to orientate data on the disk. The track can be divided into information and data blocks. The normal user has no access to information blocks. On a track an information block follows a data block and the next information block.

The most important data which appears in an information block indicates which track and block is being read and provides two checksums. The first is formed on the information block itself and the second on the following data block. These checksums are important to determine if the track contains errors.

The track gap

In addition to the information and data blocks there is a separate section which is written on disk. This is the *track gap*. The track gap contains no significant information but is required for every track.

During data writing it is important not to write new data over old data on the disk (a track is round). To prevent such overwriting, a "safety gap" between the first and last block of a disk must exist. Another reason for this gap is the fact that there is not always space for a complete block in the remaining space. This also creates a gap. The gap is about $2B8 (696) bytes long in the Amiga recording format. The number of bytes in the gap is not always the same since it can change slightly due to speed differences in the motor. The data in the gap is not important since DOS does not need it or check it.

9.3.1 Construction of block headers

Since the general construction of a track should not present any more difficulties, the information block will be examined more closely. The information block (block header) can be divided into five areas.

The beginning of the block header is formed by two sequential 0 bytes coded in MFM format. Translated, this results in two sequential $AAAA-longwords. These are followed by two standard sync markers ($44894489).

After the sync marks are four bytes, which contain information about the construction and characteristics of the track.

```
Format identification    ($FF)
Track number
Sector number
Number of sectors to the gap.
```

These four bytes, like the following data, have not yet been converted into the MFM format.

The format identification indicates - as the name implies - that the track read corresponds to the recording format of the Amiga. An MS-DOS disk is also coded in MFM format, but has a different track structure which is registered by DOS instantly through the missing format identification.

The track number indicates on which track of the disk reading just occurred. The same is true of the sector number.

The next byte indicates how many sectors exist before the track gap. The current sector is included in the count. The value one indicates that the track gap follows this sector. This byte is important since the track gap is not static (fixed location) but can exist after any sector.

The next sector contains 16 bytes which are not used by DOS. These bytes were provided to record the chaining of the blocks. These 16 bytes are free (filled with nulls) and can be used for data which is not meant to be accessible to the ordinary user (such as a serial number for the program). This area is not suitable for copy protection data.

After the 16 unused bytes is the checksum for the block header and the data block. This checksum is used by DOS for finding errors on the disk. Both sums are formed by the coded data and is also stored in MFM format. How these checksums are formed will be demonstrated in the next chapter.

Some older copy protection systems which are still in use, are based on the fact that the data of the blocks is intact, even though the checksum is in error and the block becomes unreadable for DOS. Simple copy programs cannot duplicate these programs, or correct the checksum.

When the bytes of the complete block header are calculated, the result is the following:

```
Explanation          Byte B.Nr.    Byte in MFM code
Bytes before Sync     2*   $00     00       4*    $AA
Sync-Mark              -           04       2*    $4489 (Word)
Info-Part             4*    ??     08       8*     ??
Unused Part          16*   $00     16      32*    $AA
Block-Checksum         -           48       8*     ??
Data-Checksum          -           56       8*     ??
                                      ------------------
                                           64 = $40K
```

The ?? stand for bytes whose value depends on the current block header.

After the 64 bytes of the block header comes the data block consisting of 2*512 = 1,024 ($400) bytes. In addition there are the 64 bytes of the block header, resulting in 1,088 ($440) bytes. On each track there are 11 blocks and one track gap of about $2B8 (696) bytes. This produces a total of about 12,664 (3,178) bytes.

9.3.2 Construction of the data block

The data block is much simpler in construction than the block header. It consists of two times 512K data bytes in MFM format. In the block header two adjacent MFM coded longwords form an uncoded longword. In the data block an uncoded longword is formed by two longwords, but differs from the coding of the block header in that the two coded adjacent longwords do not result in an uncoded longword. The two 512 bytes are separated.

The first and the 512th, the second and the 513th longword, etc. are combined into one uncoded longword.

9.3.3 The calculation of checksums

The checksums which are formed for the data block and the block header were discussed earlier. At this point we'll discuss the calculation of these checksums.

The checksum for a block header is calculated only for the information part and the 16 unused bytes. The sync mark isn't included. This results in a byte count of 40 ($28). For the data block the checksum is calculated for all 1,024 ($400)K. A routine is available to the operating system for calculating these checksums.

The following routine calculates the checksum for the storage area indicated. D1 contains the number of bytes for which the sum should be calculated. A0 has the pointer to the beginning of the data.

The number of bytes is divided by four to obtain the number of longwords which must be considered. The result is a number which must be a multiple of four. It is not possible to consider more than $FFFC bytes.

```
feada4 move.l  D2,-(A7)           Save D2
feada6 lsr.w   #2,D1              Number of bytes\ 4
feada8 subq.w  #1,D1              Number -1
feadaa moveq   #$00,D0            Set result to zero
feadac move.l  (A0)+,D2           Get longword
feadae eor.l   D2,D0              and attach
feadb0 dbf     D1,$feadac         branch if counter
                                  not finished
feadb4 andi.l  #$55555555,D0      remove invalid bits
feadba move.l  (A7)+,D2           Restore D2
feadbc rts                        Return jump
```

To calculate the checksum for a block header, the following program is used.

```
lea      Datastart,a0   Pointer to block header in A0
moveq    #$28,d1        Indicate number of bytes
jsr      $feada4        Calculate checksum
```

The result of the calculation is returned in D0 and can be used at the discretion of the programmer.

To calculate the data checksum, the pointer to the data block must be passed in A0 and the byte number 1,024 ($400) in D1.

9.3.4 How is a track coded?

The operating system unfortunately does not make a separate routine available for coding of a track. The user must be satisfied with a routine which codes only one block. However, this routine also calculates the checksum.

Before discussing the coding of a block or track, another often-used routine will be examined. This routine is used for coding a block header and is used often by DOS. During the call of the routine, the block header is passed in D0 and the pointer to the buffer where the coded header should be stored in A0.

```
fead46 movem.l D3-D2,-(A7)        Save D2 and D3
fead4a move.l  D0,D3              Byte to D3
fead4c lsr.l   #1,D0              Shift right
fead4e bsr.l   $fead62            Code odd bits
fead52 move.l  D3,D0              Byte to D0
fead54 bsr.l   $fead62            Code even bits
fead58 bsr.l   $feadbe            Clock bit of the next
                                  Byte corrected
fead5c movem.l (A7)+,D3-D2        Restore register
fead60 rts                        Return jump

fead62 andi.l  #$55555555,D0      Filter odd bits
fead68 move.l  D0,D2              Result to D2
fead6a eori.l  #$55555555,D2      Determine clock bits
fead70 move.l  D2,D1              Result to D1
```

```
fead72 lsl.l   #1,D2          Shift left one
fead74 lsr.l   #1,D1          Bit one to the right
fead76 bset    #31,D1         Set first bit
fead7a and.l   D2,D1          Link to sort out clock Bits
fead7c or.l    D1,D0          Set clock bits
fead7e btst    #0,-1(A0)      Determine if previous
                              Byte ended with null bit
fead84 beq.s   $fead8a        Yes, bits are right
fead86 bclr    #31,D0         Reset first bit
fead8a move.l  D0,(A0)+       Store value
fead8c rts                    Return jump
```

Correct the first clock bit of the next byte if a byte was inserted. The pointer to the next byte is in A0.

```
feadbe move.b  (A0),D0     Get byte
feadc0 btst    #0,-1(A0)   Is last bit of the previous byte set?
feadc6 bne.s   $feadd4     Yes, reset clock bit
feadc8 btst    #6,D0       Test next data bit
feadcc bne.s   $feadda     Branch if bit is set
feadce bset    #7,D0       Set clock bit
feadd2 bra.s   $feadd8     Unconditional jump
feadd4 bclr    #7,D0       Reset clock bit
feadd8 move.b  D0,(A0)     Write byte
feadda rts                 Return jump
```

Assuming the header $FF020406 was passed to the routine the individual steps of the coding are:

```
$FF240406 =>  %1111 1111 0010 0100 0000 0100 0000 0110
```

These bits are shifted to the right and at first only the odd bits are coded. This results in:

```
%0111 1111 1001 0010 0000 0010 0000 0011
```

Next the odd clock bits in this new longword are reset and then all even data bits are reversed.

```
        %0111 1111 1001 0010 0000 0010 0000 0011
AND     %0101 0101 0101 0101 0101 0101 0101 0101
        -----------------------------------------
        %0101 0101 0001 0000 0000 0000 0000 0001
EOR     %0101 0101 0101 0101 0101 0101 0101 0101
      -------------------------------------------
        %0000 0000 0100 0101 0101 0101 0101 0100
```

The purpose of this will be evident soon.

Next all reversed data bits are shifted right and left. A logical AND is then performed. The first bit of the longword shifted right is also set.

```
        %1000 0000 0010 0010 1010 1010 1010 1010
AND     %0000 0000 1000 1010 1010 1010 1010 1000
        -----------------------------------------
        %0000 0000 0000 0010 1010 1010 1010 1000
```

As a result of the logical operation only 1 bit exist where in the original longword two null bits existed. This is the case when the set clock bits must be introduced. The result of the last logical operation with the longword, in which all clock bits were removed, are ORed to obtain the final coded value:

```
        %0101 0101 0001 0000 0000 0000 0000 0001
OR      %0000 0000 0000 0010 1010 1010 1010 1000
        ----------------------------------------
        %0101 0101 0001 0010 1010 1010 1010 1001
```

A test must be performed to determine if the last data bit of the previous byte was set or reset. If it was set, the first clock bit of the longword must be reset, or it will arrive at the original longword, which was already considered in the calculation. The first coded longword is:

```
$5512AAA9
```

Analog to the coding of the odd bits is the coding of the even bits which results in the value $5524A4A4. The final coded header looks like this:

```
$5512AAA9 5524A4A4
```

After the second longword was stored, the gap between the second and following longword must be corrected. For this task a jump is made to the routine starting at $FEADBE.

The process of coding a complete track is rather time consuming. It can be speeded up considerably, at least for data blocks, by the blitter.

Before examining this routine, some foundations must be laid. A complete explanation of programming the blitter would be too lengthy for inclusion here. However, the functioning of the graphic function QBlit, which is called by the routine to be discussed, is explained. It may be puzzling why a graphic function is being used since the coding of a block does not involve graphic operations.

The QBlit function

The QBlit function just mentioned is stored in the Graphic library, but is used for more than graphic operations. During the call of the function, a pointer to a previously created structure is passed. This structure is attached to the end of a list in which there are structures used for programming the blitter. If a structure has been processed, the next one is used. When a structure is in use, the blitter can be programmed through multitasking until the control of the list is returned to the system.

QBlit has the task of waiting until the blitter is free and then passes control over it to a program. To which program control is passed, is determined in the structure.

The blitter structure which is used has the name blitNode. For the coding of a block it is passed in a slightly changed form and appears as follows:

Offset		Explanation
00	$00	Pointer to next structure.
04	$04	Pointer to program to be executed.
08	$08	Length of data for coding.
12	$0c	Pointer to Source.
16	$10	Pointer to destination.
20	$14	Content for BLTSIZE.
22	$16	Value depends on application.
26	$1A	Pointer to drive Port.

The pointer to the next structure is accepted by the QBlit function and does not have to be set by the programmer. The pointer to the program to be executed must point to the user routine to which the QBlit function jumps as soon as the blitter becomes free for this structure.

The last entry in the structure requires an explanation. It concerns the pointer to the message port of the drive which is addressed. The address of this port has been stored in the Device structure and is stored in the ORequest structure under "Unit". The pointers to the Msg. ports in the Trackdisk Device structure can be found under the following offsets:

Offset		Explanation
36	$24	Pointer to Port for Drive 0.
40	$28	Pointer to Port for Drive 1.
44	$2C	Pointer to Port for Drive 2.
48	$30	Pointer to Port for Drive 3.

If a "Drive Not Present" error is returned, the pointer is set to null.

The QBlit function performs the program indicated in the structure when its turn arrives. The return jump from QBlit occurs when the user program returns a null in D0. If this isn't the case, a branch occurs to another program through the indicated vector after the return jump from the user program. For this, the pointer is set in the first program to the next. With this linkage of programs it is possible to perform several tasks with the blitter through one call of the QBlit function. Aside from the blitter programming, several tasks can be performed by these user programs, but in most cases this does not make sense.

Since the QBlit function originated in the Graphic library, there is a pointer to the Graphic library in the Trackdisk Device structure starting at Offset 56 ($36). It may be curious, but a disk operation without the presence of the Graphic library is not possible.

The coding routine will be explained next. The information part of the block header must be passed to this routine since it's linked completely.

Since the process of coding data into the MFM format was already described in detail, a similar description of the coding by the blitter is not provided.

Coding of a block

In A0 is the pointer to the data block which is coded (Source).
In A1 is the pointer to the buffer into which the coded data is written (Destination).
In A3 is the pointer to the Msg. port of the drive.
In A6 is the pointer to the Device structure.
In D0 is the uncoded information portion of the block header.

```
feaadc movem.l A4/A2/D2,-(A7)   Save register
feaae0 move.l  A1,A4            Pointer to write buffer
feaae2 move.l  A0,A2            Pointer to data buffer
feaae4 move.l  D0,D2            Block header to D0
feaae6 moveq   #$00,D0          Clear D0
feaae8 lea     0(A4),A0         Pointer to write buffer
feaaec bsr.l   $fead46          Code null bytes and write into
                                buffer
feaaf0 move.l  #$44894489,4(A4) Store sync mark
feaaf8 move.l  D2,D0            Block header to D2
feaafa lea     8(A4),A0         Pointer to buffer
feaafe bsr.l   $fead46          Code block header and store in
                                buffer
feab02 moveq   #$03,D2          Set counter to 3
feab04 moveq   #$00,D0          Enter null bytes in buffer
feab06 bsr.l   $fead46
feab0a dbf     D2,$feab04       Branch until counter done
feab0e lea     8(A4),A0         Set pointer to coded block header
feab12 moveq   #$28,D1          Set number of bytes
feab14 bsr.l   $feada4          Calculate block checksum
feab18 lea     48(A4),A0        Pointer position in block
feab1c bsr.l   $fead46          Store checksum
feab20 move.l  #$00000200,D0    Set number of bytes
feab26 move.l  A2,A0            Pointer to data buffer
feab28 lea     64(A4),A1        Set pointer in write buffer
feab2c bsr.l   $feab4a          Code data block
feab30 lea     64(A4),A0        Pointer to the beginning of
                                coded data
feab34 move.w  #$0400,D1        Set counter
feab38 bsr.l   $feada4          Calculate checksum for data
feab3c lea     56(A4),A0        Pointer to position
feab40 bsr.l   $fead46          Store checksum
feab44 movem.l (A7)+,A4/A2/D2   Restore register
feab48 rts                      Return jump
```

Coding the data block

In A0 is the pointer to the data buffer.
In A1 is the pointer to the write buffer.
In A3 is the pointer to the Msg. port of the drive.
In A6 is the pointer to the Device structure.
In D0 is the number of data which are coded.

```
feab4a link    A2,#-30          Make space in stack
feab4e move.w  D0,D1            Number to D1
feab50 lsl.w   #2,D1            BltSize
feab52 ori.w   #$0008,D1        Determine register
feab56 move.w  D1,-10(A2)
```

```
feab5a movem.l A1-A0/D0,-22(A2)    Create structure
feab60 move.l  #$00feab9e,-26(A2)  Set function pointer
feab68 move.l  A3,-4(A2)           Pass pointer to port
feab6c lea     -30(A2),A1          Set pointer to beginning of
                                   structure
feab70 move.l  A6,-(A7)            Store A6
feab72 move.l  56(A6),A6           Get pointer to GfxBase
feab76 jsr     -276(A6)            Function QBlit
feab7a move.l  (A7)+,A6            Restore A6
feab7c bsr.l   $fea70a             Wait for Reply Msg.
feab80 movem.l -22(A2),A1-A0/D0    Restore register
feab86 move.l  D0,D1               Byte number to D1
feab88 move.l  A1,A0               Set pointer to beginning of
                                   data
feab8a bsr.l   $feadbe             Correct border
feab8e adda.l  D1,A0               Pointer to next seam
feab90 bsr.l   $feadbe             Correct border
feab94 adda.l  D1,A0               Pointer to end
feab96 bsr.l   $feadbe             Correctly border
feab9a unlk    A2                  Release stack
feab9c rts                         Return jump
```

Following is the function to which the QBlit function jumps.

In A0 the pointer to $Dff000 is passed.
In A1 is the pointer to the previously created structure.
In A6 is the pointer to the Trackdisk Device structure.

```
feab9e move.l  A5,-(A7)            Save A5
feaba0 move.l  A1,A5               Save pointer to structure
feaba2 bsr.l   $feb2cc             Set mode for A,B,D and
                                   BLTALWM
feaba6 move.l  A5,A1               Restore pointer
feaba8 movem.l 8(A1),A5/D1-D0      Get pointer to source and
                                   destination
feabae move.l  D1,76(A0)           Source to Source B
feabb2 move.l  D1,80(A0)           Source to Source A
feabb6 move.l  A5,84(A0)           Destination to Destination D
feabba move.w  #$1db1,64(A0)       Value for BLTCON0
feabc0 move.w  #$0000,66(A0)       Value for BLTCON1
feabc6 move.w  20(A1),88(A0)       Start blitter, BLTSIZE
feabcc move.l  #$00feabd8,4(A1)    Pointer to next function
feabd4 move.l  (A7)+,A5            restore A5
feabd6 rts                         Return jump
```

This is the second function. It is called when the blit process of the first has been completed. The parameters passed are the same.

```
feabd8 move.l  A5,-(A7)           Save A5
feabda movem.l 8(A1),A5/D1-D0     Values from the structure
feabe0 move.l  A5,76(A0)          Source B = Destination
feabe4 move.l  D1,80(A0)          Source A = Source
feabe8 move.l  A5,84(A0)          Destination D = Destination
feabec move.w  #$2d8c,64(A0)      Set BLTCON0
feabf2 move.w  20(A1),88(A0)      Set BLTSIZE, start
feabf8 move.l  #$00feac04,4(A1)   Store next function
feac00 move.l  (A7)+,A5           Restore A5
feac02 rts                        Return jump
```

Function 3

```
feac04 move.l  A5,-(A7)            Save A5
feac06 movem.l 8(A1),A5/D1-D0      Values from the structure
feac0c add.l   D0,D1               Pointer to end of source
feac0e subq.l  #2,D1               Set -2
feac10 adda.l  D0,A5               Pointer to end of
feac12 adda.l  D0,A5               Destination
feac14 subq.l  #2,A5               Set -2
feac16 move.l  D1,76(A0)           Source B = Source End
feac1a move.l  D1,80(A0)           Source A = Source End
feac1e move.l  A5,84(A0)           Destination D = Destination-End
feac22 move.w  #$0db1,64(A0)       Set BLITCON0
feac28 move.w  #$1002,66(A0)       Set BLITCON1
                                   (count backwards)
feac2e move.w  20(A1),88(A0)       Set BLTSIZE, start
feac34 move.l  #$00feac40,4(A1)    Next function
feac3c move.l  (A7)+,A5            Restore A5
feac3e rts                         Return jump
```

Function 4

```
feac40 move.l  A5,-(A7)            Save A5
feac42 movem.l 8(A1),A5/D1-D0      Value from structure
feac48 adda.l  D0,A5               Pointer to end of first part of
                                   destination
feac4a move.l  A5,76(A0)           Enter from Source B
feac4e move.l  D1,80(A0)           Source A = Source
feac52 move.l  A5,84(A0)           Destination D = End of the
                                   first part of destination
feac56 move.w  #$1d8c,64(A0)       Set BLTCON0
feac5c move.w  #$0000,66(A0)       Set BLTCON1
feac62 move.w  20(A1),88(A0)       Set BLTSIZE, start
feac68 move.l  #$00feac74,4(A1)    Next function
feac70 move.l  (A7)+,A5            Repeat A5
feac72 rts                         Return jump
```

Function 5

```
feac74 moveq   #$00,D0             Clear D0
feac76 move.l  D0,4(A1)            Clear function pointer
feac7a move.l  26(A1),A1           Get pointer to port
feac7e bsr.l   $fea6f2             Send Msg. (Blit-End)
feac82 moveq   #$00,D0             Set Flag for End
feac84 rts                         Return jump
```

The following routine is called by its own function. It sets the registers BLTAFWM, BLTALWM, BLTBMOD, BLTAMOD, BLTDMOD and BLTCDAT.

```
feb2cc moveq   #$00,D0             Clear D0
feb2ce lea     68(A0),A1           Pointer to BLTAFWM
feb2d2 move.l  #$ffffffff,(A1)     Set BLTAFWM, BLTALWM
feb2d8 lea     98(A0),A1           Pointer to BLTBMOD
feb2dc move.l  D0,(A1)+            Erase Mode B,A
feb2de move.w  D0,(A1)+            Erase Mode D
feb2e0 addq.l  #8,A1               Pointer to BLTCDAT
feb2e2 move.w  #$5555,(A1)         Set BLTCDAT
feb2e6 rts                         Return jump
```

The knowledge gained so far is not sufficient to write programs which codes a block with the help of the routines discussed.

The reason for this is that the Wait function is used to wait for the termination of blitter activity. This routine for waiting for the termination of blitter activity and for sending the return message, is inspected closely.

To understand the following routines, it should be mentioned that there is another Msg. Port structure (Reply port) at offset 174 ($AE) from the basic address of the Msg. Port for the drive, which is used to accept return messages.

As in the Reply Port structure, within the Msg. Port structure for the drive, there is a Message structure which is sent as message for this and other wait processes to the Reply port. The Message structure is located at offset 94 ($5E).

Then comes the routine for sending the return message (Message) to terminate the waiting process.

In A1 is the pointer to the Drive port (Msg. port).
In A6 is the pointer to the Trackdisk Device structure.

```
fea6f2 lea     174(A1),A0     Pointer to Reply port
fea6f6 lea     94(A1),A1      Pointer to Reply message
fea6fa move.l  A6,-(A7)       Save A6
fea6fc move.l  $000004,A6     Get pointer to ExexBase
fea702 jsr     -366(A6)       Call PutMsg function
fea706 move.l  (A7)+,A6       Restore A6
fea708 rts                    Return jump
```

The following routine is used to wait until the routine shown above sends a message to the Reply port.

In A3 is the pointer to the Drives port (Msg. port).
In A6 is the pointer to the Trackdisk Device structure.

```
fea70a move.l  #$00000400,D0  Signal set to D0
fea710 move.l  A6,-(A7)       Save A6
fea712 move.l  52(A6),A6      Pointer to ExecBase
fea716 jsr     -318(A6)       Wait function
fea71a move.l  (A7)+,A6       Restore A6
fea71c lea     174(A3),A0     Pointer to Reply port
fea720 move.l  A6,-(A7)       Save A6
fea722 move.l  52(A6),A6      Pointer to ExecBase
fea726 jsr     -372(A6)       GetMsg function
fea72a move.l  (A7)+,A6       Restore A6
fea72c tst.l   D0             Message arrived ?
fea72e beq.s   $fea70a        No, error, wait some more
fea730 rts                    Return jump
```

Normally all these routines are called from the Trackdisk task. For this reason the Reply port is tailored for the Trackdisk task, which means that all messages are sent to it. If the routine for coding a block or another routine which uses this waiting function is called, the message which signals the termination of the wait is sent to the Trackdisk task.

The user task which is waiting, would never obtain the message and therefore would wait forever.

To avoid this, a trick can be used. The Reply port for the user task must be informed to get the message to its destination. It is sufficient to point the *mp_SigTask entry, which points to the Trackdisk task, to the user task. This entry is at offset 16 starting at Reply port.

After termination of the user program, the pointer to the task must be pointed again to the Trackdisk task. During the change to the user task any disk accesses must be avoided because the system would get into major difficulties.

There is another matter to consider. During the processing of the user task, the Trackdisk task should not interfere. To avoid this it should be fooled into assuming that it is already executing and therefore cannot accept additional assignments. A bit in the Drive port exists for this. Setting bit 0 at offset 34 ($22) prevents the processing of messages by the Trackdisk task. Resetting the bit permits the processing to resume. The following program shows this process while coding a block with the help of the routine discussed previously. The source and destination must be added to this routine.

```
;Code.s Amiga Disk Drives Inside and Oout
Device         = 350
Port           = 36
RepPort        = 174
SigTask        = 16
Task           = 276
FindName       = -276

Header        = $FF240406

               move.l $4,a6
               lea Name,a1
               lea Device(a6),a0        ;Pointer to Devicelist
               jsr FindName(a6)         ;find Device
               tst.l d0
               beq Error
               move.l Task(a6),a0       ;Pointer to user Task
               move.l d0,a6
               move.l Port(a6),a3       ;Drives-Port-Address
               lea RepPort(a3),a1       ;Reply-Port-Address
               move.l SigTask(a1),-(a7) ;save old Pointer
               move.l a1,-(a7)          ;save Reply-Port
               move.l a0,SigTask(a1)    ;store user Task
               bset #0,34(a3)           ;set Trackdisk-Task to
                                        ;waiting position


; everything is prepared for the call of the desired routine

               move.l #Header,d0
               lea Source,a0
               lea Destination,a1
               jsr $feaadc             ;code Block
```

```
               move.l (a7)+,a1          ;restore Reply port
               move.l (a7)+,SigTask(a1) ;store old Pointer
               bclr #0,34(a3)           ;release Task again
Error:         rts                       ;Return jump

Name:          dc.b 'trackdisk.device',0

               END
```

We'll end the chapter by showing how DOS codes an entire track.

The listing which follows is only a fragment from a larger routine which also writes the track and checks for errors during writing. The writing of a track is not important at this point and will be discussed later.

In A2 is the pointer to the destination buffer (coded).
In A3 is the pointer to the Drive port.
In A5 is the pointer to the source buffer (uncoded).
In A6 is the pointer to the Trackdisk Device structure.
In D2 is the track number. The track number must be stored as a long-word.

```
          .
          .
fea112 moveq   #$0b,D4         Number of the Blocks (11)
fea114 moveq   #$00,D5         erase Block counter
fea116 move.l  #$ff000000,D0   DOS-code to D0
fea11c move.l  D5,D1           Block counter to D1
fea11e lsl.l   #8,D1           move to its Position
fea120 or.l    D1,D0           enter into Header
fea122 or.l    D4,D0           enter number of Blocks to the Gap
fea124 move.l  D2,D1           Track-Number to D1
fea126 swap    D1              bring Number into Position
fea128 or.l    D1,D0           and enter into Header
fea12a move.l  A2,A1           Destination to A1
fea12c move.l  A5,A0           Source to A0
fea12e bsr.l   $feaadc         code Block
fea132 addq.l  #1,D5           increment Block counter
fea134 adda.l  #$00000440,A2   increment Pointer to Destination
                               Buffer
fea13a adda.l  #$00000200,A5   increment Pointer to Source Buffer
fea140 subq.l  #1,D4           decrease Number of Blocks
fea142:66d2 bne.s   $fea116    branch if not done with coding
          .
          .
```

The counter for the number of the remaining blocks is also the counter for the number of blocks to the gap. If a track is coded with this or a similar program, the gap is always at the end of the track (after block 11).

9.3.5 Decoding a track

Now that we know how a track is coded, we'll learn how it can be decoded again.

A routine exists in the operating system which decodes a block. In addition another routine is used which decodes the block header. Just as in the coding of a track, first the routine for decoding of the header is described.

An uncoded longword is created from two sequential, coded longwords. A0 is the pointer to the first longword. The result is returned in D0 as an uncoded longword.

```
fead8e move.l  (A0)+,D0           get first longword
fead90 move.l  (A0)+,D1           get second longword
fead92 andi.l  #$55555555,D0      remove clock bits
fead98 andi.l  #$55555555,D1      remove clock bits
fead9e lsl.l   #1,D0              correct bits
feada0 or.l    D1,D0              and perform logical operation
feada2 rts                        Return jump
```

Starting with the example for coding of a block header, the decoding appears as follows.

You may recall that the info portion ($FF240406) of the block header was coded into two longwords ($5512AAA9 5524A4A4).

```
$5512AAA9 =  %0101 0101 0001 0010 1010 1010 1010 1001
$5524A4A4 =  %0101 0101 0010 0100 1010 0100 1010 0100
```

After getting these two longwords, the clock bits are removed.

```
       %0101 0101 0001 0010 1010 1010 1010 1001
AND    %0101 0101 0101 0101 0101 0101 0101 0101
       ----------------------------------------
       %0101 0101 0001 0000 0000 0000 0000 0001
```

and

```
       %0101 0101 0010 0100 1010 0100 1010 0100
AND    %0101 0101 0101 0101 0101 0101 0101 0101
       ----------------------------------------
       %0101 0101 0000 0100 0000 0100 0000 0100
```

Then the longword representing the odd bits (the first longword), is shifted left by one bit and a logical OR is performed with the second longword.

```
      %1010 1010 0010 0000 0000 0000 0000 0010
OR    %0101 0101 0000 0100 0000 0100 0000 0100
      ---------------------------------------
      %1111 1111 0010 0100 0000 0100 0000 0110
```

The result of the logical operation is the original header.

The decoding of data is much faster than coding, but still rather demanding if an entire track must be decoded. For this reason the blitter is used.

In A0 is the pointer to where the decoded data is stored (Destination).
In A1 is the pointer to the buffer where the data for decoding are stored (Source).
In A3 is the pointer to the Drive port.
In A6 is the pointer to the Trackdisk Device structure.
In D0 is the number of bytes to be decoded.

```
feacb2 link    A2,#-30            Make space in the stack
feacb6 movem.l A1-A0/D0,-22(A2)   Write values into the structure
feacbc lsl.w   #2,D0              calculate BLTSIZE
feacbe ori.w   #$0008,D0
feacc2 move.w  D0,-10(A2)         and store
feacc6 move.l  #$00feacf0,-26(A2) Pointer to function
feacce move.l  A3,-4(A2)          store pointer to port
feacd2 lea     -30(A2),A1         Pointer to start of structure
feacd6 move.l  A6,-(A7)           save A6
feacd8 move.l  56(A6),A6          get pointer to GfxBase
feacdc jsr     -276(A6)           call QBlit function
feace0 move.l  (A7)+,A6           restore A6
feace2 bsr.l   $fea70a            wait for Reply-Msg.
feace6 movem.l -22(A2),A1-A0/D0   restore register
feacec unlk    A2                 correct stack
feacee rts                        Return jump
```

Now the listing of the function which is called by the QBlit function:

In A0 is the pointer to the beginning of the custom chips ($DFF000).
In A1 is the pointer to the BlitNode structure which was created.
In A6 is the pointer to the Trackdisk Device structure.

```
feacf0 move.l  A5,-(A7)         Save A5
feacf2 move.l  A1,A5            save pointer to structure
feacf4 bsr.l   $feb2cc          set values for blitter
                                see previous chapter
feacf8 move.l  A5,A1            get pointer to structure
feacfa movem.l 8(A1),A5/D1-D0   get pointer to source,
                                destination
                                and Length
fead00 adda.l  D0,A5            determine end address of source
                                for odd Bits
fead02 subq.l  #1,A5            decrease by one
fead04 move.l  A5,80(A0)        enter in Source A
fead08 adda.l  D0,A5            End address of even bits
fead0a move.l  A5,76(A0)        enter in Source B
fead0e add.l   D0,D1            End address destination
fead10 subq.l  #1,D1            -1
fead12 move.l  D1,84(A0)        enter Destination D
```

```
fead16 move.w  #$1dd8,64(A0)     set BLTCON0
fead1c move.w  #$0002,66(A0)     set BLTCON1 (count backwards)
fead22 move.w  20(A1),88(A0)     set BLTSIZE, Start
fead28 move.l  #$00fead34,4(A1)  enter new function
fead30 move.l  (A7)+,A5          restore A5
fead32 rts                       Return jump
```

Function 2

```
fead34 moveq   #$00,D0           clear D0
fead36 move.l  D0,4(A1)          erase pointer to function
fead3a move.l  26(A1),A1         get pointer to port
fead3e bsr.l   $fea6f2           Put message
fead42 moveq   #$00,D0           clear D0 (End Flag)
fead44 rts                       Return jump
```

To decode a block, the following program is required. The source and the destination must be inserted.

```
;Decode.s Amiga Disk Drives Inside and Out
Device          = 350
Port            = 36
RepPort         = 174
SigTask         = 16
Task            = 276
FindName        = -276

Number    = $200

                move.l $4,a6
                lea Name,a1
                lea Device(a6),a0        ;Pointer to Devicelist
                jsr FindName(a6)         ;find Device
                tst.l d0
                beq Error
                move.l Task(a6),a0       ;Pointer to user Task
                move.l d0,a6
                move.l Port(a6),a3       ;Drive-Port-Address
                lea RepPort(a3),a1       ;Reply-Port-Address
                move.l SigTask(a1),-(a7) ;save old Pointer
                move.l a1,-(a7)          ;save Reply-Port
                move.l a0,SigTask(a1)    ;enter user Task
                bset #0,34(a3)           ;set Trackdisk-Task to
                                         ;wait position

;everything is ready to call the desired routine
                lea Source,a1
                lea Destination,a0
                move.l #Anzahl,d0
                jsr $feacb2              ;decode Block

                move.l (a7)+,a1          ;restore Reply-Port
                move.l (a7)+,SigTask(a1) ;enter old Pointer
                bclr #0,34(a3)           ;release Task again
Error:         rts                       ;Return jump

Name:           dc.b 'trackdisk.device',0

END
```

9.4 The disk registers

The coding and decoding of data as discussed in the preceding sections have provided you with a basic working knowledge. The most important step is the reading and writing of data to and from the disk. To be able to understand these processes completely a working knowledge of the hardware registers is needed. We'll discuss this in the following sections.

9.4.1 The Drive Status register

This register is used to check to see if a disk is in the drive, if it is write protected, etc.

Register Drive-Status = $BFE001

Port		Name	Description
CIAA	PA5	DSKRDY*	The bit indicates if the drive is ready to accept commands Ready => Bit = 0
CIAA	PA4	DSKTRACK0*	The position of the head of the addressed drive is on Head at Null => Bit = 0
CIAA	PA3	DSKPROT*	Indicates if the disk which is in the drive, is write protected. Protected => Bit = 0
CIAA	PA2	DSKCANGE*	Indicates if a disk is in the drive. Disk in Drive =>Bit = 1 The bit is actuated then the stepmotor is moved.

Bits are valid only for the drive indicated by the Drive Select register. This register is discussed next.

If several drives are addressed at the same time, the bits are set according to events when one of the drives is in the condition to display. For example the DSKCANGE bit would be LOW, when one of the addressed drives does not have a disk present.

Example for waiting for disk ready:

```
L1:     BTST #5,$BFE001
        BNE.S L1
        RTS
```

9.4.2 The Drive Select register

With this register the four different drives are addressed and a selection is made if the upper (head 0) or lower side (head 1) of the disk should be addressed. In addition, the step motor of the selected drive is controlled through this register.

Register drive select = $BFD100

Port	Pin	Name	Description
CIAB	PB7	DSKMOTOR*	This bit controls the drive motors of the four drives. If the bit is LOW, while a drive is selected, its motor switches on. Additional description follows.
CIAB	PB6	DSKSEL3*	Bit for Drive 3 Drive addressed => Bit = 0
CIAB	PB5	DSKSEL2*	Bit for Drive 2 Drive addressed => Bit = 0
CIAB	PB4	DSKSEL1*	Bit for Drive 1 Drive addressed => Bit = 0
CIAB	PB3	DSKSEL0*	Bit for Drive 0 Drive addressed => Bit = 0
CIAB	PB2	DSKSIDE*	Indicates which head was selected. Head 0 (bottom) => Bit = 1
CIAB	PB1	DISDIREC	Indicates if the head of the drives should move inward or outward. Inward => Bit = 0
CIAB	PB0	DISKSTEP*	The motor is moved with this bit. During a change from HIGH to LOW, the head moves in the direction indicated.

A closer explanation requires the assignment of individual drive motor conditions. If a drive is selected for a certain activity by resetting the corresponding DSKSEL bit, this stores the condition of the DSKMOTOR bit (switches the motor on or off). The Motor command is considered until the corresponding drive bit again switches from HIGH to LOW. When this happens, the condition of the DSKMOTOR bit is transmitted to the motor. The DISKSTEP bit should always be reset to HIGH after a change from HIGH to LOW or problems develop when changing drives.

It is possible to address several drives simultaneously and to move the heads of both drives at the same time. Here are some programming examples which show how the motors of the individual drives are addressed.

Switch on motor for drive 0, select head 1 (down):

```
MotorOn:  MOVE.B #$7D,$BFD100      All Drive bits on HIGH
          NOP
          NOP
          MOVE.B #$75,$BFD100      Drive 0, Motor on
          MOVE.W #$B000,D0
L1:       DBRA D0,L1               Waiting loop
          RTS
```

Switch off motor for drives 0 and 1:

```
MotorOff: MOVE.B #$FD,$BFD100      All drive bits on HIGH
          NOP
          NOP
          MOVE.B #$E7,$BFD100      Drives 0 and 1 off
          MOVE.W #$B000,D0
L1:       DBRA D0,L1               Waiting loop
          RTS
```

The following demonstrates how the head of one or more drives can be moved inward or outward.

Move head one track inward:

```
HeadIn:   BCLR #1,$BFD100
          BCLR #0,$BFD100
          BSET #0,$BFD100
          MOVE.W #3000,D0
L2:       DBRA D0,L2          Waiting loop
L1:       BTST #5,$BFE001     Wait for Disk-Ready
          BNE.S L1
          RTS
```

Move head one track to outside:

```
HeadOut:  BSET #1,$BFD100
          BCLR #0,$BFD100
          BSET #0,$BFD100
          MOVE.W #3000,D0
L2:       DBRA D0,L2          Waiting loop
L1:       BTST #5,$BFE001     Wait for Disk-Ready
          BNE.S L1
          RTS
```

Since these routines do not work with the operating system, they must lock out the interrupts before use or make the task of the drive "think" that it is already working. How this is done was discussed in the examples for coding and decoding.

9.4.3 The Disk LEN and Disk Pointer register

To read or write data, the hardware must be informed from which memory region this should occur. In addition it must know how many bytes are processed.

To set the start address, two connecting registers (DSKPTH and DSKPTL) exist. Only 19 bits are accepted for determining the address. With these 19 bits the lowest 512 Kbytes of memory can be addressed (2^19 => 512 Kbytes).

The DSKPT registers don't have to be set individually. This can be done with an assignment.

```
lea     Pointer,a0
move.l  a0,DSKPTH
```

The two registers have the addresses $DFF020 and $DFF022.

After the start address has been passed, the length is passed to start the DMA access. Both tasks can be performed with the DSKLEN register.

This register can be divided into two areas. First it controls the DMA access, and also the number of words to be transferred are passed to it. There are 13 bits available for passing the word length. With the 13 bits a maximum of 16 Kbytes can be transferred.

Register DSKLEN = $DFF024

Pin	Name	Description
15	DMAEN	Enable Disk-DMA. DMA enable => Bit = 1
14	WRITE	Indicates if read or write. Read => Bit = 0
13-0	LENGTH	Number of words to be transferred.

To start Disk DMA without errors, a certain procedure must be followed. The register must be written twice with the same value and only then is data transferred. To avoid errors during the transmission, the DMAEN bit should be erased before and after the transmission.

Starting the transmission appears as follows:

1. Set the register to $4000, to block the DMA.
2. Write the desired value into the register.
3. Write the same value again to start the DMA.
4. After termination of the transmission the register is set again to $4000 to prevent writing on the disk.

If the transmission is started, the DSKLEN register is decremented and the DSKPT register is incremented. When the counter in the SKLEN register is zero, the transmission stops.

Because of a hardware error, the last three bits of the data to be transmitted to disk are ignored. Also the last word which should be written from the disk to storage is not sent. Therefore the user must always transmit one word more than the required data.

9.4.4 The Disk Byte Read register

This register is also a Status register, but indicates other results than the Drive Status register. In addition the last 8 bits can be used to read the byte which is currently being read from the disk. If a byte has arrived, the byteready bit is set to HIGH. When the register is read, the byteready bit is reset automatically.

Register DSKBYTR = $DFF01A

Bit	Name	Description
15	BYTEREADY	Indicates when a byte has arrived from the disk. The bit is erased during a read access.
14	DAMON	Indicates if the Disk-DMA is permitted. Both the bit in the DSKLEN register, and the DMACON register must be set.
13	DISKWRITE	Indicates if the read or write modeis switched on in the DSKLEN register.
12	WORDEQUEL	Indicates that the controller has found a sync mark. The bit is set only as long as the sync mark is recognized (about 2 Microseconds).
11-8		Not used.
7-0	DATA	Contains the data just read from the disk.

9.4.5 The ADKCON and ADKCONR registers

The ADKCON and ADKCONR registers are important parts of the Amiga disk controller. ADKCON is the write address and ADKCONR the read address of the registers. Not all bits of these registers are used for disk accesses. The lower 8 bits are used for music programming.

Register ADKCON = $DFF09E, ADKCONR = $DFF010

Bit	Name	Explanation
15	CLR/SET	Detailed explanation follows.
14	PRECOMP1	Upper bit of the bit pair PRECOM1 and 2.
13	PRECOMP2	Lower bit of the bit pair for the pre-compensation during writing: Value 00 => 00 ns Value 01 => 140 ns Value 10 => 280 ns Value 11 => 560 ns
12	MFMPREC	Value 0 selects MFM-Format. Value 1 selects GCR-Format.
11	UARTBRK	Value 1 => Output to Paula. Value 0 => Output to Disk.
10	WORDSYNC	Switch on synchronization for a certain word. The word must be in $DFF07E. The transmission of the data starts only after the mark is found. Value 1 => Synchronization on
9	MSBSYNC	Switch on synchronization for GCR-Sync-Marking. Value 1 => Synchronization on
8	FAST	Switch on the writing speed: Value 1 => MFM-Format, 2 ms per Bit Value 0 => GCR-Format, 4 ms per Bit
7-0		Not for Disk Controller.

The CLR/SET bit (bit 15) indicates if the bits which are set in the value to be written are set or reset in the register. If the CLR/SET bit is set, all register bits are set in the word which is written. If the CLR/SET bit is reset, all bits which are set in the word in the register, are reset.

Two examples make this clear:

```
Word to be written:    1 000 1001 0110 0000
Old Register content:  0 110 0110 1100 0000
                       --------------------
New Register content:  1 110 1111 1110 0000

Word to be written:    0 110 0010 0111 0000
Old Register content:  0 110 1111 1110 0000
                       --------------------
New Register content:  0 000 1101 1000 0000
```

9.4.6 The Disk Sync register

Register Address = $DFF07E

The controller synchronizes itself according to the word in this register, if the wordsync bit in the ADKCON register is set. The data transmission is not started before this word is found on disk.

Finding a word can trigger an interrupt (Priority 6). Further discussions about the interrupt can be found in the proper chapter.

9.4.7 The DSKDAT registers

These registers are a pair of which one (DSKDAT) is write only and the other (DSKDATR) is read only register.

Register DSKDAT = $DFF026
Register DSKDATR = $DFF008

The register serves as a data buffer during the data transmission from or to the disk with the DMA.

9.5 Reading a track

Up to now the discussion about programming the registers has been theoretical. In this chapter it is demonstrated how a track is read and decoded and how the operating system performs this task.

First the documentation of the read routine which is easy to understand if the register descriptions are used.

```
fea524 lea     132(PC)(=$fea5aa),A1   Pointer to routine
                                       for setting of DSKLEN
fea528 movem.l A4/A2/D2,-(A7)         save Register
fea52c move.l  A0,A2                  Load address to A2
fea52e move.l  D0,D2                  Number of Bytes to be read
fea530 move.l  A1,A4                  set Pointer to Routine
fea532 bsr.l   $feaddc                Routine for Motor organization
fea536 move.b  65(A3),$bfd100         set Motor Bits
fea53e move.w  #$4000,$dff024         prepare DSKLEN-Register to Read
fea546 move.l  #$000003e8,D0          Value for Waiting loop
fea54c bsr.s   $fea4f0                Wait until Drive is finished
fea54e lea     $dff000,A1             Pointer to Custom-Chips
fea554 move.l  A2,32(A1)              set DSKPT-Register
fea558 move.w  #$1002,156(A1)         block Sync-Interrupt
fea55e move.w  #$8002,154(A1)         release Disk-Block-Ready-Int.
fea564 btst    #2,$bfe001             Disk in the Drive?
fea56c bne.s   $fea572                branch if Disk present
fea56e moveq   #$1d,D2                Error Message to D2
fea570 bra.s   $fea59e                End
fea572 lsr.w   #1,D2                  convert number of Bytes
                                      into number of words
fea574 ori.w   #$8000,D2              set Bit for DMA permitted
fea578 move.l  D2,D0                  Value to d0
fea57a jsr     (A4)                   set DSKLEN-Register, Start
fea57c bsr.l   $fea70a                wait for return message
fea580 lea     $dff000,A1             Pointer to Custom-Chips
fea586 move.w  #$0002,154(A1)         block Disk-Block-Int.
fea58c move.w  #$4000,36(A1)          block Disk-DMA (DSKLEN)
fea592 btst    #2,$bfe001             Disk removed ?
fea59a beq.s   $fea56e                yes, Error => End
fea59c moveq   #$00,D2                Error-Flag on ok
fea59e bsr.l   $feae42                organize Motor
fea5a2 move.l  D2,D0                  Return message to D0
fea5a4 movem.l (A7)+,A4/A2/D2         restore Register
fea5a8 rts                            Return jump
```

Routine for setting the DSKLEN registers. The value is in D0.

```
fea5aa move.w  D0,36(A1)              set Register
fea5ae move.w  D0,36(A1)              start DMA
fea5b2 rts                            Return jump
```

Next follows a program which reads a track with the help of the previously discussed routine into the indicated buffer.

```
;Read.s Amiga Disk Drives Inside and Out
Device          = 350
Port            = 36                    ;Offset for Drive 0
RepPort         = 174
SigTask         = 16
Task            = 276
FindName        = -276

Number          = $397c
Track           = 20

            move.l $4,a6         ;get ExecBase
            lea Name,a1          ;set Pointer to Name
            lea Device(a6),a0    ;Pointer to Device-List
            jsr FindName(a6)     ;seek Trackdisk-Device
            tst.l d0             ;Device found ?
            beq Error            ;No, error
            move.l Task(a6),a0   ;get Pointer to user Task
            move.l d0,a6         ;Pointer to Task to A6
            move.l Port(a6),a3   ;get Pointer to Drives-
                                 ;Port (Drive 0)
            lea RepPort(a3),a1   ;Pointer to RepPort
            move.l SigTask(a1),-(a7);save Pointer to Task
            move.l a1,-(a7)      ;save Pointer to RepPort
            move.l a0,SigTask(a1) ;enter user Task
            bset #0,34(a3)       ;block Trackdisk-Task

; From here on starts the actual load routine

            move.l #1,d0              ;Value for Motor on
            jsr $fea462               ;switch Motor on
            move.l #Track,d0          ;Track-Number to D0
            move.w #Track,74(a3)
            jsr $fea3da              ;move Head to position
            move.l 78(a3),a0         ;Pointer to read buffer
            move.l #Number,d0        ;Number of bytes to read
                                     ;to D0
            jsr $fea524              ;read Track
            clr.l d0                 ;Value for Motor off
            jsr $fea462              ;turn Motor off

; End of the load routine. The modified pointers must be
; restored again.

            bclr #0,34(a3)            ;release Task again
            move.l (a7)+,a1           ;get Pointer to Task
            move.l (a7)+,SigTask(a1) ;store again in Port
Error:        rts                     ;Return jump

Name:           dc.b 'trackdisk.device',0


            END
```

You may have noticed that in the routine the synchronization on a certain bit combination ($4489) was not switched on. For this reason the data read was not synchronized. A special routine must be called which retroactively recognizes the sync mark and corrects the data read.

The routine for control of the drive motor erases the previously set bits. Using special tricks (see the description of the RAW commands) the routine can be made to wait for the sync before the start of read. For the sake of clarity the following illustrates how this would be done for a user load routine.

```
move.w #$8400,$dff09e       Switch on the synchronization
move.w #$4498,$dff07e       determine after which word
                            synchronization should occur.
```

The routine which corrects the unsynchronized data is explained in the following section.

The main jump is at $FEAFE2. The previous routine is called by the main routine.

The first routine searches for the block header. It is found even if it is not known if the block header has clock or data bits in the beginning of the buffer. Even if the data is shifted by several bits (because they are not synchronized), the header is found.

The routine orients itself on the four $AA bytes which always precede the sync mark. Even if the data is shifted a $AAAA or $5555 is found. If the routine finds a $5555, it knows that it found a clock bit.

In A2 is the pointer to the data to be decoded.
In D0 is the number of the bytes to be searched.

```
feaf4c movem.l A2/D4-D2,-(A7)    save Register
feaf50 move.w  #$aaaa,D3         first control value to D3
feaf54 move.w  #$5555,D4         second control value to D4
feaf58 move.l  A0,A2             Pointer to data to A2
feaf5a adda.l  D0,A2             determine end of the data
feaf5c move.w  (A0)+,D2          Word to D2
feaf5e cmp.w   D3,D2             Word = first control value?
feaf60 beq.s   $feaf98           branch, if word was found
feaf62 cmp.w   D4,D2             Word = second control value?
feaf64 beq.s   $feaf70           branch, if Word was found
feaf66 cmpa.l  A0,A2             End of data reached?
feaf68 bhi.s   $feaf5c           continue if not reached
feaf6a moveq   #$ff,D0           Error message to D0
feaf6c move.l  D0,A0             D0 to A0
feaf6e bra.s   $feaf92           Return jump with error message
```

Jump if the second word was found. This means that a clock bit was found first.

```
feaf70 moveq    #$0f,D0       Counter for the Number of
                              Bits shifted
feaf72 lea      $feafa2,A1    Pointer to Table to A1
```

```
feaf78 cmpa.l  A0,A2         End of Data?
feaf7a bls.s   $feaf6a       yes, error
feaf7c move.w  (A0)+,D1      next Word to D1
feaf7e cmp.w   D2,D1         still the $AA Byte before the Sync-
                             Mark ?
feaf80 beq.s   $feaf78       yes, get next Word
feaf82 subq.l  #2,A0         let A0 point to previous Word
feaf84 move.l  (A0),D1       get Longword
feaf86 cmp.l   (A1)+,D1      is this a Sync-Mark?
feaf88 beq.s   $feaf90       branch, if possible Sync
feaf8a subq.l  #2,D0         else decrement number of bits
feaf8c bge.s   $feaf86       branch when not counted down
feaf8e bra.s   $feaf5c       else, no Sync found, continue search
feaf90 subq.l  #4,A0             Pointer to beginning of Sync-
feaf92 movem.l (A7)+,A2/D4-D2    restore Register
feaf96 rts                       Return jump
```

Set pointer if a data bit was found first.

```
feaf98 moveq   #$0e,D0       Counter for for number of  shifted
                             Bits to D0
feaf9a lea     $feafc2,A1    set Pointer to table
feafa0 bra.s   $feaf78       unconditional Jump
```

Table for sync recognition when a clock bit was found first.

```
feafa2: 2244 a244 4891 2891 5224 4a24 5489 1289
feafb2: 5522 44a2 5548 9128 5552 244a 5554 8912
```

Table for sync recognition when a data bit was found first.

```
feafc2: 9122 5122 a448 9448 a912 2512 aa44 8944
feafd2: aa91 2251 aaa4 4894 aaa9 1225 4489 4489
```

The main routine which corrects the data read, begins here.

In A2 is the pointer to the data buffer in which the data is stored starting at Offset 1664 ($680).
In A3 is the pointer to the Drive port.
In A6 is the pointer to the Trackdisk Device structure.

```
feafe2 movem.l A2/D6-D2,-(A7)   save Register
feafe6 link    A4,#-16          make space in Stack
feafea move.l  78(A3),A2        Pointer to Load Buffer
feafee lea     1664(A2),A2      Pointer to beginning of data
feaff2 move.l  A2,A0            Pointer to data to A0
feaff4 addq.l  #2,A0
feaff6 move.l  #$00000abc,D0    Bytes read exceed Counter
feaffc bsr.l   $feaf4c          search for Block-Header
feb000 cmpa.l  #$ffffffff,A0    Header found ?
feb006 beq.l   $feb1e8          branch if not found
feb00a move.l  A0,D5            Pointer to Header to D5
feb00c move.l  D0,D2            Number of Bits to shift
feb00e addq.l  #8,A0            set pointer to longword behind
                                sync
feb010 moveq   #$09,D4          Counter for checksum creation
feb012 moveq   #$00,D6          clear register for checksum
feb014 tst.l   D2               do bits have to shift?
feb016 bne.s   $feb03c          branch if yes
```

```
feb018 move.l  (A0),-8(A4)        else, store Header Bytes
feb01c move.l  4(A0),-4(A4)       store Header Bytes

feb022 move.l  #$55555555,D1      Value for filtering of clock bits
feb028 move.l  (A0)+,D0           get Longword
feb02a and.l   D1,D0              filter out Clock Bits
feb02c eor.l   D0,D6              form Checksum
feb02e dbf     D4,$feb028         decrement Counter
feb032 move.l  (A0)+,-16(A4)      store Header-Checksum
feb036 move.l  (A0),-12(A4)       store Header-Checksum
feb03a bra.s   $feb070            unconditional Jump
```

Jump to here if the data was shifted and stored.

```
feb03c bsr.l   $feb204            get header shifted properly
feb040 move.l  D0,-8(A4)          store Header
feb044 bsr.l   $feb204            get Header shifted properly
feb048 move.l  D0,-4(A4)          store Header
feb04c move.l  D5,A0              Sync-Address to A0
feb04e addq.l  #8,A0              Pointer to Header
feb050 bsr.l   $feb204            get Longword
feb054 andi.l  #$55555555,D0      filter Clock Bits
feb05a eor.l   D0,D6              form Checksum
feb05c dbf     D4,$feb050         branch if not finished
feb060 bsr.l   $feb204            get Header-Checksum
feb064 move.l  D0,-16(A4)         and store
feb068 bsr.l   $feb204            get Header-Checksum
feb06c move.l  D0,-12(A4)         and store
feb070 lea     -16(A4),A0         Pointer to header checksum
feb074 bsr.l   $fead8e            decode Checksum
feb078 cmp.l   D0,D6              compare with calculated sum
feb07a bne.l   $feb1ec            branch if not equal, error
feb07e lea     -8(A4),A0          set Pointer to Header
feb082 bsr.l   $fead8e            decode Header
feb086 move.l  D0,D3              Decoded Header to D3
feb088 move.l  D3,-(A7)           store in Stack
feb08a cmpi.b  #$ff,0(A7)         is DOS identification right?
feb090 bne.l   $feb1ec            branch if error
feb094 move.b  1(A7),D0           get Track number from Header
feb098 cmp.b   75(A3),D0          right Track?
feb09c bne.l   $feb1ec            branch if error
feb0a0 move.l  (A7)+,D3           nonsense since header is already
                                   in D3
feb0a2 moveq   #$00,D0            clear D0
feb0a4 move.b  D3,D0              Number of Blocks to Gap
feb0a6 mulu    #$0440,D0          calculate number of Bytes to Gap
feb0aa move.l  D5,A0              Pointer to Sync-Address to A0
feb0ac move.l  A2,A1              Pointer to beginning of data in
                                  buffer Destination during copying
feb0ae move.l  D2,D1              Number of Bits to shift
feb0b0 move.l  D0,D4              Distance to Gap to D4
feb0b2 bsr.l   $feb214            copy data properly
```

The $FEA214 routine copies the data located up to the track gap in the data buffer where the right data starts (offset 1664 ($680). If the data is shifted by a few bits, they are corrected. This task is performed by the blitter.

```
feb0b6 moveq   #$00,D2           clear D2
feb0b8 move.b  D3,D2             Number of Blocks to the Gap
feb0ba subi.l  #$0000000b,D2     -Maximum Number-1
feb0c0 neg.l   D2                Number of Blocks after the Gap
feb0c2 beq.s   $feb0ee           Branch if no additional
feb0c4 add.l   D4,D5             determine Address of Gap
feb0c6 move.l  #$0000067c,D0     Number of Bytes to be searched-
feb0cc move.l  D5,A0              Address of the Gap to A0
feb0ce addq.l  #2,A0
feb0d0 bsr.l   $feaf4c           search Sync for Gap
feb0d4 cmpa.l  #$ffffffff,A0     Sync found ?
feb0da beq.l   $feb200           no, error
feb0de move.l  D0,D1             Number of Bits to b shifted
feb0e0 move.l  A2,A1             Data Buffer to A1
feb0e2 adda.l  D4,A1             determine destination address for
                                 Data
feb0e4 move.l  D2,D0             Number of Blocks after Gap
feb0e6 mulu    #$0440,D0         determine Bytes after Gap
feb0ea bsr.l   $feb214           copy Bytes with help of Blitter
feb0ee move.l  A2,A0             Data Buffer address to A0
feb0f0 adda.l  D4,A0             calculate the Address of newly
                                  copied data
feb0f2 bsr.l   $feadbe           correct borders (see coding in
                                 MFM-Format)
feb0f6 lea     11968(A2),A0      Pointer to end of data
feb0fa move.w  #$aaa8,D0         Value for End marking
feb0fe btst    #0,-1(A0)         test last data Bit
feb104 beq.l   $feb10c           branch if Bit was reset
feb108 bclr    #15,D0            else erase End marking
feb10c move.w  D0,(A0)           store Mark
feb10e move.w  #$aaaa,(A2)       store mark of beginning
feb112 moveq   #$00,D4           set Block counter to zero
feb114 moveq   #$0b,D2           Number of Blocks to 11
feb116 move.w  D3,D5             first Header to D5
feb118 lsr.w   #8,D5             shift off Number of Blocks to Gap
```

From here on starts the part which tests the properly shifted track for errors.

```
feb11a cmpi.l  #$2aaaaaaa,0(A2,D4.W)  found beginning ?
feb122 beq.s   $feb130                branch when found
feb124 cmpi.l  #$aaaaaaaa,0(A2,D4.W)  found beginning ?
feb12c bne.l   $feb1f0                branch when error
feb130 cmpi.l  #$44894489,4(A2,D4.W)  found Sync ?
feb138 bne.l   $feb1f0                branch when error
feb13c lea     8(A2,D4.W),A0     Pointer to Header
feb140 moveq   #$28,D1           Number of Bytes for Checksum
feb142 bsr.l   $feada4           form Sum
feb146 move.l  D0,D6             store Sum
feb148 lea     48(A2,D4.W),A0    Pointer to Checksum in Header
feb14c bsr.l   $fead8e           decode Checksum
feb150 cmp.l   D6,D0             Checksum OK ?
feb152 bne.l   $feb1f8           branch if error
feb156 lea     8(A2,D4.W),A0     Pointer to Header
feb15a bsr.l   $fead8e           decode Header
feb15e move.l  D0,-(A7)          store Header
feb160 cmpi.b  #$ff,0(A7)        DOS identification ok?
feb166 bne.l   $feb1f4           branch if error
feb16a move.b  1(A7),D1          Track-Number to D1
feb16e cmp.b   75(A3),D1         Track-Number ok?
feb172 bne.l   $feb1f4           branch if error
```

```
feb176 move.b  2(A7),D1         Sector number to D1
feb17a cmp.b   D5,D1            Sector number ok?
feb17c bne.l   $feb1f4          branch, if error
feb180 move.b  D2,3(A7)         enter number of Sectors until Gap
feb184 move.l  (A7)+,D0         restore Header
feb186 lea     8(A2,D4.W),A0    Pointer to Header-Position
feb18a bsr.l   $fead46          enter Header again
feb18e lea     8(A2,D4.W),A0    Pointer to Header-Position
feb192 moveq   #$28,D1          Counter for Checksum
feb194 bsr.l   $feada4          calculate Checksum
feb198 lea     48(A2,D4.W),A0   Pointer to Sum entry
feb19c bsr.l   $fead46          enter Checksum again
feb1a0 lea     64(A2,D4.W),A0   Pointer to beginning of
                                Data Blocks
feb1a4 move.w  #$0400,D1        Number of Bytes to be calculated
feb1a8 bsr.l   $feada4          calculate Sum
feb1ac move.l  D0,D6            store Sum
feb1ae lea     56(A2,D4.W),A0   Pointer to Data Checksum
feb1b2 bsr.l   $fead8e          decode Sum
feb1b6 cmp.l   D6,D0            compare with calculated Sum
feb1b8 bne.l   $feb1fc          branch, if error
feb1bc subq.l  #1,D2            decrement Counter for number of
                                Blocks
feb1be addq.b  #1,D5            increment Block number
feb1c0 cmpi.b  #$0b,D5          Block number = 11
feb1c4 blt.s   $feb1c8          branch, if not 11
feb1c6 moveq   #$00,D5          else Block number = Null
feb1c8 addi.w  #$0440,D4        Pointer to next Block
feb1cc cmpi.w  #$2ec0,D4        all Blocks checked ?
feb1d0 bne.l   $feb11a          no, next Block
feb1d4 move.l  D3,D1            first Header in Buffer to D1
feb1d6 lsr.l   #8,D1            Track-Number to lowest Byte
feb1d8 moveq   #$00,D0          clear D0
feb1da move.b  D1,D0            Track-Number to D0
feb1dc unlk    A4               release Stack
feb1de movem.l (A7)+,A2/D6-D2   restore Register
feb1e2 rts                      Return jump
```

Passing the error numbers, return jump.

```
feb1e4 nop
feb1e6 bra.s   $feb1dc
feb1e8 moveq   #$15,D0
feb1ea bra.s   $feb1e4     Return jump
feb1ec moveq   #$1b,D0
feb1ee bra.s   $feb1e4     Return jump
feb1f0 moveq   #$16,D0
feb1f2 bra.s   $feb1e4     Return jump
feb1f4 moveq   #$17,D0
feb1f6 bra.s   $feb1e4     Return jump
feb1f8 moveq   #$18,D0
feb1fa bra.s   $feb1e4     Return jump
feb1fc moveq   #$19,D0
feb1fe bra.s   $feb1e4     Return jump
feb200 moveq   #$1a,D0
feb202 bra.s   $feb1e4     Return jump
```

The number of blocks in the buffer is returned in D0.

The next routine is quite useful. It reads a track, removes the gap and tests the track for errors. If an error occurrs, it tries again to read the track until the number of possible recovery attempts has been exhausted.

The track number to be read is located at offset 74 in the Drives Port structure. In addition the track number must also be stored at the beginning of the buffer to be read. Bit 0 at offset 2 starting at the buffer must also be erased.

The pointer to the buffer into which the data should be written is located at offset 78 in the Drive Port structure. After loading, they are located starting at offset $1668 ($684).

```
fea99e movem.l A2,-(A7)        save A2
fea9a2 move.l  78(A3),A2       get Pointer to Buffer
fea9a6 moveq   #$01,D0         Value for Motor on
fea9a8 bsr.l   $fea462         switch Motor on
fea9ac moveq   #$00,D0         clear D0
fea9ae move.w  74(A3),D0       Track-Number to D0
fea9b2 bsr.l   $fea3da         Head positioning
fea9b6 lea     1668(A2),A0     Pointer to beginning of data
fea9ba move.w  #$397c,D0       Number of Bytes to be read
fea9be bsr.l   $fea524         read Track
fea9c2 tst.l   D0              Error during read ?
fea9c4 beq.s   $fea9ce         branch, if no error
fea9c6 move.b  D0,3(A2)        else pass error number
fea9ca bra.l   $fea9f8         unconditional Jump
fea9ce bsr.l   $feafe2         correct Track, remove Gap
fea9d2 move.b  D0,3(A2)        store Sector number
fea9d6 cmpi.b  #$0b,D0         compare Sector number with 11
fea9da bcs.s   $fea9f8         branch, if everything is ok
fea9dc addq.b  #1,66(A3)       else error occurred
fea9e0 move.b  66(A3),D0       increment number of errors
fea9e4 cmp.b   52(A3),D0       Number of errors at maximum?
fea9e8 bgt.s   $fea9f8         branch, if maximum
fea9ea andi.b  #$03,D0         Error number a multiple of 4?
fea9ee bne.s   $fea9b6         no, try again
fea9f0 move.w  #$ffff,76(A3)   else value for new positioning
                               of Head (first to Null)
fea9f6 bra.s   $fea9ac         unconditional Jump
fea9f8 movem.l (A7)+,A2        restore A2
fea9fc rts                     Return jump
```

Finally this routine which reads a track, decodes it and moves it to the buffer. The destination address must be indicated by the program. The program uses the routine just described.

```
;Read-d.d Amiga Disk Drives Inside and Out
Device          = 350
Port            = 36          ;Offset for Drive 0
RepPort         = 174
SigTask         = 16
Task            = 276
FindName        = -276

Track           = 20          ;Number of Track to be read
```

```
;Destination          = ????
;Destination must be provided by the user.
Destination    =$5000        ; for testing only
            move.l $4,a6           ;get ExecBase
            lea Name,a1            ;set Pointer to Name
            lea Device(a6),a0  ;Pointer to Device-List
            jsr FindName(a6) ;search for Trackdisk-Device
            tst.l d0             ;Device found ?
            beq Error            ;No, Error
            move.l Task(a6),a0  ;get Pointer to user Task
            move.l d0,a6         ;Pointer to Task to A6
            move.l Port(a6),a3  ;get pointer to drive
                                 ;Port (Drive 0)
            lea RepPort(a3),a1  ;Pointer to RepPort
            move.l SigTask(a1),-(a7);save Pointer to Task
            move.l a1,-(a7)    ;save Pointer to RepPort
            move.l a0,SigTask(a1)    ;store user Task
            bset #0,34(a3)       ;block Trackdisk-Task
; The actual load routine starts here

            move.l #Track,d0    ;Track-Number to D0
            move.w d0,74(a3)    ;store in Structure
            move.l 78(a3),a2    ;Pointer to Data buffer
            move.w d0,(a2)      ;store Track-Number
            bclr #0,2(a2)       ;erase Bit 0
            clr.b 66(a3)        ;erase Errornumber
            jsr $fea99e         ;read Track
            clr.l d0            ;Value for Motor off
            jsr $fea462         ;switch Motor off
            move.b 3(a2),d0    ;first Block number to D0
            cmp.b #$0b,d0       ;Number larger than 11
            bcc Ende            ;yes, Error
            move.b #$0b,d6      ;Sector Number
            clr.l d0            ;first Block = Null
            sub.b 3(a2),d0
            bpl l1
            addi.b #$0b,d0      ;Address of the Block
l1:         mulu #$440,d0       ;calculate Null
            lea 1664(a2),a4     ;Pointer start of Data
            adda.l d0,a4        ;Pointer to Block zero
            lea Destination,a5 ;set Pointer to dest
            clr.l d7            ;start at Sector zero
l3:         lea 64(a4),a1       ;Pointer to Data block
            move.l a5,a0        ;Destination to A0
            move.l #$200,d0     ;Number  to be decoded
            jsr $feacb2         ;decode Data
            adda.l #$200,a5     ;increment Dest Pointer
            sub.b #1,d6        ;decrement number of Blocks
            beq Ende            ;branch, when done
            add.b #1,d7         ;increment Block number
            cmp.b 3(a2),d7      ;continue start of Buffer
            bne l2              ;no, continue normally
            lea 1664(a2),a4     ;Pointer start of Buffer
            bra l3              ;unconditional Jump
l2:         add.l #$440,a4      ;Pointer to next Block
            bra l3              ;unconditional Jump
```

```
; End of the load routine. The modified pointers
; must be restored.

Ende:       bclr #0,34(a3)        ;release Task again
            move.l (a7)+,a1       ;get Pointer to Task
            move.l (a7)+,SigTask(a1) ;enter again in Port
Error:        rts                    ;Return jump

Name:         dc.b 'trackdisk.device',0


            END
```

With the help of the routines described and the examples, you should now be able to read and decode a track directly.

9.6 Writing a track to disk

The coding of data and how it is written on a desired track will now be shown.

The Write routine is similar in construction to the Load routine. One difference is that during the writing of a track the write density changes, depending on its position. The operating system only uses two of the four possible write densities. A change in write density is performed starting at track 80.

The Save routine of the operating system demonstrates how a track is written.

```
fea5b4 lea      -12(PC)(=$fea5aa),A1    Pointer to DSKLEN register
fea5b8 movem.l A4/A2/D2,-(A7)           save Register
fea5bc move.l   A0,A2                   Pointer to Write buffer
fea5be move.l   D0,D2                   Number Bytes to be written
fea5c0 move.l   A1,A4                   Pointer to DSKLEN-Routine
fea5c2 bsr.l    $feaddc                 Motor control
fea5c6 move.b   65(A3),$bfd100          set Motor Bits
fea5ce move.w   #$4000,$dff024          block Disk-DMA
fea5d6 move.l   #$000003e8,D0           Value for Waiting loop
fea5dc bsr.l    $fea4f0                 Wait
fea5e0 lea      $dff000,A1              Pointer to Custom-Chips
fea5e6 move.l   A2,32(A1)               set DSKPT-Register
fea5ea move.w   #$1002,156(A1)          clear Disk block interrupt
                                        Request
fea5f0 move.w   #$8002,154(A1)          enable Disk-Block-Int.
fea5f6 btst     #2,$bfe001              Disk in Drive?
fea5fe bne.s    $fea606                 branch, if ok
fea600 moveq    #$1d,D2                 Error number to D2
fea602 bra.l    $fea6e2                 End, Error
fea606 btst     #3,$bfe001              Disk write protected?
fea60e beq.l    $fea6ee                 branch, if protected
fea612 move.l   $0004,A0                Pointer to ExecBase
fea616 move.w   #$4000,$dff09a          Disable-
fea61e addq.b   #1,294(A0)              Macro
fea622 btst     #4,34(A6)               Bit 4 in Status-Byte set
fea628 beq.s    $fea642                 branch, if reset
fea62a moveq    #$23,D2                 else Error number to D2
fea62c move.l   $0004,A0                get ExecBase
fea630 subq.b   #1,294(A0)              Enable-
fea634 bge.s    $fea63e
fea636 move.w   #$c000,$dff09a          Macro
fea63e bra.l    $fea6e2                 unconditional Jump
fea642 bset     #5,34(A6)               set Bit 5
fea648 move.l   $0004,A0                ExecBase to A0
fea64c subq.b   #1,294(A0)              Enable-
fea650 bge.s    $fea65a
fea652 move.w   #$c000,$dff09a          Macro
fea65a move.w   #$6000,158(A1)          reset Bit for lesser write
density
fea660 move.w   76(A3),D0               Track-Position
```

```
fea664 move.w  #$8000,D1          Write density to 0
fea668 cmp.w   38(A3),D0          compare if writing with
                                  another write density
fea66c bls.s   $fea686            Track number smaller
fea66e move.w  #$a000,D1          else Write density 1
fea672 cmp.w   40(A3),D0          compare Track-Number
fea676 bls.s   $fea686            Track-Number smaller
fea678 move.w  #$c000,D1          else Write density 2
fea67c cmp.w   42(A3),D0          compare Track-Number
fea680 bls.s   $fea686            Track-Number smaller
fea682 move.w  #$e000,D1          else Write density 3
fea686 move.w  D1,158(A1)         store Write density
fea68a lsr.w   #1,D2              change number of Bytes to
                                  be written to Words
fea68c ori.w   #$c000,D2          set Bits for writing
fea690 move.l  D2,D0              Value to D0
fea692 jsr     (A4)               write DSKLEN-Register
                                  (see Load-Routine)
fea694 bsr.l   $fea70a            wait until Track has been
                                  written
fea698 lea     $dff000,A1         Pointer to Custom-Chips
fea69e move.w  #$0002,154(A1)     block Disk-Block-Int.
fea6a4 move.l  #$000007d0,D0      Value for time loop
fea6aa bsr.l   $fea4f0            Wait
fea6ae move.w  #$4000,36(A1)      block Disk-DMA
fea6b4 move.b  34(A6),D0
fea6b8 bclr    #5,34(A6)          erase Bit 5
fea6be btst    #4,D0
fea6c2 beq.s   $fea6d4            in user System,
                                  unconditional Jump
fea6c4 lea     120(A6),A1
fea6c8 move.l  A6,-(A7)
fea6ca move.l  20(A1),A6
fea6ce jsr     -30(A6)
fea6d2 move.l  (A7)+,A6

fea6d4 btst    #2,$bfe001         Disk in Drive?
fea6dc beq.l   $fea600            branch, if no
fea6e0 moveq   #$00,D2            return message ok
fea6e2 bsr.l   $feae42            Motor control
fea6e6 move.l  D2,D0              Return message to D0
fea6e8 movem.l (A7)+,A4/A2/D2     restore Register
fea6ec rts                        Return jump
```

Jump point when write protected.

```
fea6ee moveq   #$1c,D2            Error number to D2
fea6f0 bra.s   $fea6e2            unconditional Jump
```

We've already seen how the operating system codes a track (in the section on coding). Next a program that performs all the tasks.

The following program makes it possible to store data which is located in chip memory; ($0000-$80000). The routine has the advantage, compared with the operating system, that it does not check if there is an error on the track to be written. Therefore all disks can be written with this program, even if they are (for example) MS-DOS disks. If an error does occur, it is not intercepted by the program. A test for errors must be added.

If an error occurs, the error number is passed in D0. Otherwise D0 returns a null.

The beginning of the program is the same as the previous examples.

```
;Write.s Amiga Disk Drives Inside and Out
Device          = 350
Port            = 36          ;Offset for Drive 0
RepPort         = 174
SigTask         = 16
Task            = 276
FindName        = -276

Track           = 20          ;Number of the Tracks to read
Address         = $50000      ;The Address which memory
; starts can only be in Chip-Memory.
;Here $50000 was selected.
            move.l $4,a6            ;get ExecBase
            lea Name,a1             ;set Pointer to Name
            lea Device(a6),a0       ;Pointer to Device-List
            jsr FindName(a6) ;search for Trackdisk-Device
            tst.l d0                ;Device found ?
            beq Error               ;No, Error
            move.l Task(a6),a0  ;get Pointer to user Task
            move.l d0,a6            ;Pointer to Task to A6
            move.l Port(a6),a3      ;get Pointer to Drive-
                                    ;Port (Drive 0)
            lea RepPort(a3),a1         ;Pointer to RepPort
            move.l SigTask(a1),-(a7);save Pointer to Task
            move.l a1,-(a7)        ;save Pointer to RepPort
            move.l a0,SigTask(a1)     ;store user Task
            bset #0,34(a3)           ;block Trackdisk-Task

; Starting here is the actual write routine

            move.l #1,d0           ;Value for Motor on
            jsr $FEA462            ;switch Motor on
            move.l #TRACK,D2       ;Track-Number to D2
            move.l D2,D0           ;Track-Number to D0
            jsr $FEA3DA            ;Head positioning
            lea Address,A5         ;Pointer to Start
                       ;address of Data to transmit to A5
            move.l $52(A3),A2      ;Pointer to Write buffer
            lea 4(A2),A2
            move.w #$FFF,D0        ;erase counter value
            move.l #$AAAAAAAA,D1 ;Value for erasing
L1:         move.l D1,(A2)+        ;erase Write buffer
            dbra D0,L1             ;branch, until done
            movea.l  $52(A3),A2  ;Pointer to Write buffer
            lea $680(A2),A2 ;Pointer to beginning of data
            moveq #$0B,D4          ;Number of Blocks
            moveq #0,D5            ;Block counter to Null
L2:         move.l #$FF000000,D0 ;DOS id of the Header
            move.l D5,D1           ;Block number to D1
            lsl.l #8,D1            ;shift Number to proper
```

```
                                    ;position
            or.l D1,D0              ;store in Header
            or.l D4,D0         ;store number of blocks to gap
            move.l D2,D1            ;Track-Number to D1
            swap D1              ;Number to proper Position
            or.l D1,D0              ;enter Track-Number
            movea.l A2,A1           ;Write buffer to A1
            movea.l A5,A0           ;Pointer to Data to A0
            jsr $FEAADC            ;code Data and shift into
                                    ;Write buffer
            addq.l #1,D5            ;increment Block counter
            adda.l #$440,A2         ;Pointer to next Block
                                    ;in the Write buffer
            adda.l #$200,A5         ;Pointer to next Data
            subq.l #1,D4         ;decrement number of Blocks
            bne.s L2      ;branch, if not finished coding
            movea.l $52(A3),A0     ;Pointer to Write buffer
            lea 4(A0),A0
            move.w #$353E,D0        ;Data number for writing
            jsr $FEA5B4             ;write Track
            move.l #0,d0            ;Value for Motor off
            jsr $FEA462             ;switch Motor off

; End of the write routine. The changed
; Pointers must be restored.

Ende:       bclr #0,34(a3)             ;release Task again
            move.l (a7)+,a1            ;get Pointer to Task
            move.l (a7)+,SigTask(a1) ;store in Port again
Error:      rts                        ;Return jump

Name:       dc.b 'trackdisk.device',0

            END
```

9.7 The disk interrupts

During disk operations it is possible to trigger two different kinds of interrupts. One can be triggered when the controller recognizes a sync mark. This is triggered through a flag line from the CIAB and produces a Level 6 interrupt. It is used by the operating system only with the RAW commands.

The second interrupt is triggered, when a data block is completely transmitted. This occurs during reading and writing of a track. This produces a Level 1 interrupt.

First the Disk Block interrupt is discussed since it is the most important for the system. When transmission to or from the disk is started with the DSKLEN register, the Save and Load routines branch to a subroutine which puts the task into a wait status. This routine appears as follows:

In A3 is the pointer to the Drive port.
In A6 is the pointer to the Trackdisk Device structure.

```
fea70a move.l  #$00000400,D0        set Signal Bits
fea710 move.l  A6,-(A7)             save A6
fea712 move.l  52(A6),A6            Pointer to ExecBase
fea716 jsr     -318(A6)             Wait-Function
fea71a move.l  (A7)+,A6             restore A6
fea71c lea     174(A3),A0           Pointer to Reply-Port
fea720 move.l  A6,-(A7)             save A6
fea722 move.l  52(A6),A6            Pointer to ExecBase
fea726 jsr     -372(A6)             GetMsg-Function
fea72a move.l  (A7)+,A6             restore A6
fea72c tst.l   D0                   Msg. arrived
fea72e beq.s   $fea70a              branch, if not arrived
fea730 rts                          Return jump
```

Looking at the routine the question arises where the message is sent to restart the task. The answer is simple. As soon as the block is transmitted, the Disk Block interrupt is executed which processes the routine.

In A1 is the pointer to the Disk Resource structure.

This pointer is in the Interrupt Vector structure in the ExecBase structure.

```
fc4a80 move.w  #$0002,$dff09c         Reset the Int.-Request-Bit
fc4a88 move.l  34(A1),D0              Pointer to Reply-Msg.
fc4a8c beq.s   $fc4ac0                Guru, when not set
fc4a8e move.l  D0,A1                  Reply-Msg. to A1
fc4a90 movem.l 34(A1),A5/A1           Pointer to Interrupt-Vector-
                                      Structure
```

```
                                A1 = Pointer to Drive-Port
                                A5 = $FEA6F2
fc4a96 jmp      (A5)            Location for Jump
```

In A1 is the pointer to the Drive port.

```
fea6f2 lea      174(A1),A0      Pointer to Reply-Port
fea6f6 lea      94(A1),A1       Pointer to Reply-Message
fea6fa move.l   A6,-(A7)        save A6
fea6fc move.l   $000004,A6      Pointer to ExecBase
fea702 jsr      -366(A6)        PutMsg-Function
fea706 move.l   (A7)+,A6        restore A6
fea708 rts                      Return jump
```

With this system computer time is provided for other tasks which the Trackdisk task cannot use, since it has to wait for the completion of the disk operation.

Now to discuss an interrupt which is not important to the operating system but can occur when the disk controller synchronizes. The interrupt jumps to the following routine.

In A1 is a pointer to the Disk Resource structure.

```
fc4a98 move.w #$1000,$dff09c    erase Int.-Request-Bit
fc4aa0 move.l  34(A1),D0        Pointer to Rep-Msg.
fc4aa4 beq.s   $fc4ac0          Guru, if Msg. not present
fc4aa6 move.l  D0,A1            Pointer to Message to A1
fc4aa8 movem.l 56(A1),A5/A1     Pointer to Interrupt-Vector-
                                Structure in the
                                Drives-Port-Structure
fc4aae jmp     (A5)             Location for Jump
```

The Interrupt Vector structure addressed in the listing is not initialized. If necessary this can be used for other purposes.

The Reply Message structure is a structure which is inside the Message Port structure for the various drives (Drives port). Starting at offset 242 and 264 are the Interrupt Vector structures for the two interrupts which can be used as desired.

Appendices

Appendix A
The Diskmon.s program

```
;Diskmon.s, run from CLI only
;See the Abacus book Amiga Disk Drives Inside and Out
;Section 6.1 for instructions
;Assemble with AssemPro Amiga

Exodus:
OldOpenLibrary      =-408
CloseLibrary        =-414
AllocMem            =-198
FreeMem             =-210
Read                = -42
Write               = -48
Open                = -30
Close               = -36
FindTask            =-294
OpenDevice          =-444
CloseDevice         =-450
DoIO                =-456


        tst.l d0                ;parameter length
        beq.s run
        cmp.l #5,d0
        bne.s run
        cmp.b #"d",(a0)         ;test for "dfx:"
        bne.s run
        cmp.b #"f",1(a0)
        bne.s run
        cmp.b #":",3(a0)
        bne.s run
        move.b 2(a0),d0
        sub.b #"0",d0
        move.b d0,device        ;SET DRIVE
run:    move.l 4,a6             ;dos.library open
        lea dosname,a1
        jsr OldOpenLibrary(a6)
        move.l d0,dosbase
        beq error

        move.l #$10002,d1       ;512 byte buffer in chipmem
        move.l #512,d0          ;reserved
        jsr AllocMem(a6)
        move.l d0,buffer
        beq error

        sub.l a1,a1             ;task for trackdisk.device
        jsr FindTask(a6)
```

```
        lea diskport,a0
        move.l d0,16(a0)
        clr.l d0
        move.b device,d0        ;trackdisk.device for dfx: open
        moveq #0,d1
        lea diskioreq,a1
        lea trkdisk,a0
        jsr OpenDevice(a6)
        tst.l d0
        bne nodrive             ;close library,release memory
        move.b device,d0        ;ldrive in command line set
        add.b #"0",d0
        move.b d0,drive
        move.l dosbase,a6       ;open raw-window
        move.l #title,d1
        move.l #1005,d2
        jsr Open(a6)
        move.l d0,wdhd          ;windowhandle
        beq error

        jsr crsroff             ;cursor off
        move.l wdhd,d1          ;menu display
        move.l #top,d2
        move.l #toplen,d3
        jsr Write(a6)

        jsr dumpblock           ;output block # and $,clr errors
        move.b #"r",key         ;simlulate read command
        move.b #"h",display   ;hex display
        bra.s start

main:   jsr dumptype            ;typ output
        jsr dumpcheck           ;checksum output
        jsr getkey              ;get next key
start:  cmp.b #$1b,key          ;program end
        beq.s quit
        cmp.b #"r",key          ;read block and display
        beq readsec
        cmp.b #"w",key          ;write blockand display
        beq writesec
        cmp.b #"c",key          ;data checksum create
        beq check
        cmp.b #"#",key          ;decimal block input and read
        beq blockedit
        cmp.b #"$",key          ;hexadecimal block input and read
        beq blockedithex
        cmp.b #"+",key          ;block +1 read
        beq up
        cmp.b #"-",key          ;block -1 read
        beq down
        cmp.b #"a",key          ;ascii input

        beq asciiedit
        cmp.b #"h",key          ;hex input
        beq hexedit
        bne main
```

```
quit:   move.l dosbase,a6        ;close window
        move.l wdhd,d1
        jsr Close(a6)

        move.l 4,a6              ;close trackdisk.device
        lea diskioreq,a1
        jsr CloseDevice(a6)

nodrive:move.l buffer,a1         ;free buffer
        move.l #512,d0
        jsr FreeMem(a6)

        move.l dosbase,a1        ;close dos.library
        jsr CloseLibrary(a6)
        clr.l d0
        rts

error:  moveq #100,d0            ;returncode 100 for system error
        rts

dumphex:cmp.b #"a",display       ;ascii input- NTSC added
        beq hexstop              ;NTSC added
        move.b #"0",row          ;cursor pos.
        move.b #"6",row+1
        move.b #"0",col
        move.b #"2",col+1
        move.l buffer,buffptr    ;buffer pointer to start
        clr.w adr
        moveq #15,d6             ;16 lines
poshex: jsr cursor               ;address output
        move.w adr,d0
        jsr convword
        jsr printword
        move.b #" ",key          ;space printed
        jsr printkey
        add.w #$20,adr           ;inc. address by $20
        moveq #15,d5             ;16 words per line
        lea linebuf,a2           ;buffer for line
x:      move.l buffptr,a1
        add.l #2,buffptr         ;buffer pointer2
        move.w (a1),d0           ;get word
        jsr convword             ;convert to ascii
        move.l mytext,(a2)+      ;copy to line buffer
        dbra d5,x
        jsr printline
        move.b row+1,d0          ;cursor pos. line +1
        cmp.b #"9",d0
        bne.s l1
        add.b #1,row
        move.b #"0"-1,row+1
l1:     add.b #1,row+1
        dbra d6,poshex
hexstop: rts
```

```
convdez:lea mytext,a0               ;convert word in d0 by 4
        divu #1000,d0               ;decimal number
        add.b #"0",d0
        move.b d0,(a0)+
        clr.w d0
        swap d0
        divu #100,d0
        add.b #"0",d0
        move.b d0,(a0)+
        clr.w d0
        swap d0
        divu #10,d0
        add.b #"0",d0
        move.b d0,(a0)+
        clr.w d0
        swap d0
        add.b #"0",d0
        move.b d0,(a0)+
        rts

convword:                           ;convert word in d0 to ascii text
        moveq #3,d2
        lea mytext+4,a0
l0:     move.b d0,d1
        and.b #$0f,d1
        lsr.w #4,d0
        cmp.b #$09,d1
        bgt.s hex
        add.b #"0",d1
        bra.s do
hex:    add.b #"a"-10,d1
do:     move.b d1,-(a0)
        dbra d2,l0
        rts

printword:                          ;text output
        move.l wdhd,d1
        move.l #mytext,d2
        moveq #4,d3
        jsr Write(a6)
        rts

dumpasc:cmp.b #"h",display            ;check for hex display NTSC only
        beq ascstop
        move.b #"0",row             ;block ascii output Pal=2
        move.b #"6",row+1           ;PAl =3
        move.b #"0",col             ;PAL =0
        move.b #"2",col+1           ;Pal =2
        move.l buffer,buffptr       ;
        clr.w adr                   ;
        moveq #7,d6                 ; PAL=7
posasc: jsr cursor                  ;address output
        move.w adr,d0
        jsr convword
        jsr printword
        move.b #" ",key             ;space printed
```

```
        jsr printkey
        add.w #$40,adr
        moveq #63,d5
        lea linebuf,a2          ;line buffer
        move.l buffptr,a1
y:      move.b (a1)+,d0         ;get byte and mask ascii
        cmp.b #" ",d0
        blt.s dot
        cmp.b #"z",d0
        bgt.s dot
        move.b d0,(a2)+
        bra.s chr
dot:    move.b #".",(a2)+       ;replace control char with "."
chr:    dbra d5,y
        move.l a1,buffptr
        jsr printline           ;line buffer output
        move.b row+1,d0         ;cursor pos. line +1
        cmp.b #"9",d0
        bne.s l2
        add.b #1,row
        move.b #"0"-1,row+1
l2:     add.b #1,row+1
        dbra d6,posasc
        move.l wdhd,d1             ;NTSC only to clear 8 hex lines
        move.l #clrhex,d2          ;NTSC only
        move.l #clrhexlen,d3       ;NTSC only
        jsr Write(a6)              ;NTSC only


ascstop:rts

printline:                      ;line buffer output
        move.l wdhd,d1
        move.l #linebuf,d2
        moveq #64,d3
        jsr Write(a6)
        rts

dumpcheck:                      ;checksum output
        move.b #"0",row
        move.b #"4",row+1
        move.b #"5",col
        move.b #"1",col+1
        jsr cursor
        move.l buffer,a0        ;upper word
        move.w 20(a0),d0
        jsr convword
        jsr printword
        move.l buffer,a0        ;lower word
        move.w 22(a0),d0
        jsr convword
        jsr printword
        rts

dumpblock:                      ;block number dez. and hex. output
        move.b #"0",row
```

```
        move.b #"4",row+1
        move.b #"0",col
        move.b #"9",col+1
        jsr cursor
        move.w block,d0         ;set offset for read/write
        mulu #512,d0
        move.l d0,offset
        clr.l d0
        move.w block,d0
        jsr convdez             ;convert block decimal
        jsr printword
        move.b #"0",row         ;convert block hex.
        move.b #"4",row+1
        move.b #"1",col
        move.b #"5",col+1
        jsr cursor
        move.w block,d0
        jsr convword
        jsr printword

        move.l #clear,d4        ;clr error message
        moveq #clrlen,d5
        jsr doerr
        rts

dumptype:
        move.b #"0",row         ;block type output
        move.b #"2",row+1
        move.b #"0",col
        move.b #"2",col+1
        jsr cursor
        move.l wdhd,d1
        move.l #unkn,d2         ;typ unknown
        moveq #10,d3            ;typ-lenght
        cmp.w #2,block          ;boot       block=0,1
        bge.s noboot
        move.l #boot,d2
        bra.s noknown
noboot: move.l buffer,a0
        cmp.l #8,(a0)           ;data       1.LW=$00000008
        bne.s nodata
        move.l #dat,d2
nodata: cmp.l #$10,(a0)         ;filelist   1.LW=$00000010
        bne.s nolisting
        cmp.l #-3,508(a0)       ;filelist 127.LW=$fffffffd
        bne.s nolisting
        move.l #flist,d2
nolisting: cmp.l #2,(a0)        ;root,userdir,filehead
                                ;           1.LW=$00000002
        bne.s noknown
        cmp.l #1,508(a0)        ;root     127.LW=$00000001
        bne.s noroot
        move.l #root,d2
noroot: cmp.l #2,508(a0)        ;userdir  127.LW=$00000002
        bne.s noudir
        move.l #udir,d2
```

```
noudir: cmp.l #-3,508(a0)       ;filehead 127.LW=$fffffffd
        bne.s noknown
        move.l #fhead,d2
noknown:jsr Write(a6)
        rts

getkey: move.l wdhd,d1          ;wait for key
        move.l #key,d2          ;get next key
        moveq #1,d3
        jsr Read(a6)
        rts

printkey:                       ;char key printed
        move.l wdhd,d1
        move.l #key,d2
        moveq #1,d3
        jsr Write(a6)
        rts

cursor: jsr cursoff             ;cursor off, no status change
        move.l wdhd,d1          ;cursor on #address position
        move.l #adrpos,d2
        moveq #7,d3
        jsr Write(a6)
        lea mytext,a0
        move.l #"    ",(a0)
        btst #0,crsrstatus      ;when cursor off,clr address
        beq.s noadr
        move.l buffptr,d0       ;else output address
        sub.l buffer,d0         ;address=pointer-start
        jsr convdez
noadr:  jsr printword
        move.l wdhd,d1          ;position cursor on $address
        move.l #adrpos2,d2
        moveq #7,d3
        jsr Write(a6)
        lea mytext,a0
        move.l #"    ",(a0)
        btst #0,crsrstatus      ;when cursor off, clr address
        beq.s noadr2
        move.l buffptr,d0       ;else output address
        sub.l buffer,d0
        jsr convword
noadr2: jsr printword
        move.l wdhd,d1          ;cursor position
        move.l #pos,d2
        moveq #7,d3
        jsr Write(a6)
        btst #0,crsrstatus
        beq.s no
        jsr curson              ;cursor on. no status change
no:     rts

crsron: bset #0,crsrstatus      ;switch cursor on
curson: move.l wdhd,d1
        move.l #con,d2
```

```
        moveq #3,d3
        jsr Write(a6)
        rts

crsroff:bclr #0,crsrstatus        ;switch cursor off
cursoff:move.l wdhd,d1
        move.l #coff,d2
        moveq #4,d3
        jsr Write(a6)
        rts

doerr:  move.l dosbase,a6         ;output error message d4/d5
        move.b #"0",row
        move.b #"4",row+1
        move.b #"6",col
        move.b #"0",col+1
        jsr cursor
        move.l wdhd,d1
        move.l d4,d2
        move.l d5,d3
        jsr Write(a6)
        lea diskioreq,a0          ;clr error status
        clr.l 32(a0)
        rts

mtroff: move.l 4,a6               ;switch motor off
        lea diskioreq,a1
        move.w #9,28(a1)          ;motor on
        clr.l 36(a1)
        jsr DoIO(a6)
        move.l dosbase,a6
        rts

readsec:move.l 4,a6               ;read block into buffer
        lea diskioreq,a1
        move.w #14,28(a1)         ;test is  disk inserted
        jsr DoIO(a6)
        lea diskioreq,a1
        tst.l 32(a1)
        beq.s dsk

        move.l #nderr,d4          ;output error
        moveq #ndlen,d5
        jsr doerr
        bra here                  ;read finished

dsk:    lea diskioreq,a1          ;block read
        move.w #2,28(a1)
        move.l #512,36(a1)
        move.l buffer,40(a1)
        move.l offset,44(a1)
        jsr DoIO(a6)
        tst.l d0                  ;test for read error
        beq.s noerr

        move.l #rderr,d4          ;error output
```

```
        moveq #rdlen,d5
        jsr doerr
        bra.s here

noerr:  jsr dumpblock           ;output block number,clr error
here:   jsr mtroff
        jsr dumpcheck           ;checksum
        jsr dumptype            ;type
        cmp.b #"a",key          ;ascii input  NTSC added
        jsr dumpasc             ;               NTSC added

        cmp.b #"h",key          ;hex input    NTSC added

        jsr dumphex             ;hex. output
;       jsr dumpasc             ;ascii output orginal pal
        bra main

writesec:                       ;write block to disk
        move.l 4,a6
        lea diskioreq,a1
        move.w #14,28(a1)       ;test if disk inserted
        jsr DoIO(a6)
        lea diskioreq,a1
        tst.l 32(a1)
        beq.s dsk2

        move.l #nderr,d4        ;error output
        moveq #ndlen,d5
        jsr doerr
        bra here2

dsk2:   lea diskioreq,a1
        move.w #15,28(a1)       ;test for write-protect
        jsr DoIO(a6)
        lea diskioreq,a1
        tst.l 32(a1)
        beq.s dsk3

        move.l #pterr,d4        ;error output
        moveq #ptlen,d5
        jsr doerr
        bra here2

dsk3:   move.w #3,28(a1)        ;block write
        move.l #512,36(a1)
        move.l buffer,40(a1)
        move.l offset,44(a1)
        jsr DoIO(a6)
        lea diskioreq,a1
        move.w #4,28(a1)        ;update disk
        move.l #512,36(a1)
        move.l buffer,40(a1)
        move.l offset,44(a1)
        jsr DoIO(a6)
        tst.l d0                ;test for write error
        beq.s noerr2
```

```
        move.l #wrerr,d4        ;error output
        moveq #wrlen,d5
        jsr doerr
        bra.s here2

noerr2: jsr dumpblock           ;output everything
here2:  jsr mtroff
        jsr dumpcheck
        jsr dumphex
        jsr dumpasc
        bra main

check:  move.l buffer,a0        ;calculate buffer checksum
        moveq #126,d0
        clr.l d1
adck:   cmp.w #121,d0           ;jump over checksum
        bne.s ck
        add.l #4,a0
ck:     sub.l (a0)+,d1
        dbra d0,adck
        move.l buffer,a0        ;record checksum
        move.l d1,20(a0)
        jsr dumpcheck           ;output
        jsr dumphex
        jsr dumpasc
        bra main

blockedit:                      ;input block number in dec.
        move.b #"0",row
        move.b #"4",row+1
        move.b #"0",col
        move.b #"9",col+1
        jsr cursor
        jsr crsron
        moveq #3,d4             ;4 chars
        lea mytext,a5
in:     jsr getkey
        cmp.b #"0",key
        blt.s in
        cmp.b #"9",key
        bgt.s in
        jsr printkey
        move.b key,(a5)+        ;in text buffer
        dbra d4,in
        jsr crsroff
        clr.w block             ;convert text buffer to hex
        lea mytext,a0
        clr.w d0
        move.b (a0)+,d0
        sub.w #"0",d0
        mulu #1000,d0
        add.w d0,block
        clr.w d0
        move.b (a0)+,d0
        sub.w #"0",d0
```

```
        mulu #100,d0
        add.w d0,block
        clr.w d0
        move.b (a0)+,d0
        sub.w #"0",d0
        mulu #10,d0
        add.w d0,block
        clr.w d0
        move.b (a0)+,d0
        sub.w #"0",d0
        add.w d0,block
        cmp.w #1759,block     ;compare with last block
        bgt blockedit         ;new input
        jsr dumpblock
        bra readsec           ;read blockand display

blockedithex:                   ;input block in hex
        move.b #"0",row
        move.b #"4",row+1
        move.b #"1",col
        move.b #"5",col+1
        jsr cursor
        jsr crsron
        lea mytext,a5
        moveq #3,d4             ;4 char
retry:  jsr getkey
        cmp.b #"0",key
        blt retry
        cmp.b #"f",key
        bgt retry
        cmp.b #"9",key
        ble.s h0
        cmp.b #"a",key
        bge.s h0
        bra.s retry
h0:     jsr printkey
        move.b key,(a5)+        ;write in text buffer
        dbra d4,retry
        jsr crsroff
        move.b mytext,d0        ;convert text to hex in nibbles
        cmp.b #"9",d0
        bgt.s h1
        sub.b #"0"-"a"+10,d0
h1:     sub.b #"a"-10,d0
        lsl.b #4,d0
        move.b d0,block
        move.b mytext+1,d0
        cmp.b #"9",d0
        bgt.s h2
        sub.b #"0"-"a"+10,d0
h2:     sub.b #"a"-10,d0
        or.b d0,block
        move.b mytext+2,d0
        cmp.b #"9",d0
        bgt.s h3
        sub.b #"0"-"a"+10,d0
```

```
h3:     sub.b #"a"-10,d0
        lsl.b #4,d0
        move.b d0,block+1
        move.b mytext+3,d0
        cmp.b #"9",d0
        bgt.s h4
        sub.b #"0"-"a"+10,d0
h4:     sub.b #"a"-10,d0
        or.b d0,block+1
        cmp.w #1759,block       ;compare with last block
        bgt blockedithex
        jsr dumpblock           ;block output
        bra readsec           ;read block and display

up:     cmp.w #1759,block       ;read next block and display
        beq main
        add.w #1,block
        jsr dumpblock
        jmp readsec

down:   tst.w block             ;previous block read and disp.
        beq main
        sub.w #1,block
        jsr dumpblock
        jmp readsec

asciiedit:  move.b #"a",display   ;NTSC added
        jsr dumpasc           ;ascii input in buffer
        move.b #"0",row         ;PAL =2
        move.b #"6",row+1       ;PAl =3
        move.b #"0",col         ;PAL =0
        move.b #"7",col+1       ;PAL =7
        move.l buffer,buffptr
        jsr crsron
        jsr cursor
getasc: jsr getkey

        cmp.b #$9b,key          ;compare with cursor sequence
        bne nocurs
        jsr getkey
        cmp.b #$44,key          ;left
        beq ascleft
        cmp.b #$43,key          ;right
        beq ascright
        cmp.b #$41,key          ;up
        beq ascup
        cmp.b #$42,key          ;down
        beq ascdown
        bra.s getasc

ascright:                       ;cursor right or start of line
        cmp.b #"7",col
        blt.s csright
        cmp.b #"0",col+1
        blt.s csright
        cmp.b #"3",row
```

```
        blt csdown
        cmp.b #"0",row+1
        blt csdown
        bra getasc              ;cursor in lower left

csright:cmp.b #"9",col+1          ;set cursor
        bne.s m3
        move.b #"0"-1,col+1
        add.b #1,col
m3:     add.b #1,col+1
        add.l #1,buffptr          ;set buffer pointer
        jsr cursor
        bra getasc

csdown: cmp.b #"9",row+1          ;see above
        bne.s m2
        move.b #"0"-1,row+1
        add.b #1,row
m2:     add.b #1,row+1
        move.b #"0",col
        move.b #"7",col+1
        add.l #1,buffptr
        jsr cursor
        bra getasc

ascdown:cmp.b #"3",row            ;cursor down if possible
        blt rowdown
        cmp.b #"0",row+1
        blt rowdown
        bra getasc

rowdown:cmp.b #"9",row+1
        bne.s m4
        move.b #"0"-1,row+1
        add.b #1,row
m4:     add.b #1,row+1
        add.l #$40,buffptr        ;buffer pointer next line
        jsr cursor
        bra getasc

ascleft:cmp.b #"0",col            ;cursor left or end of line
        bgt.s csleft
        cmp.b #"7",col+1
        bgt.s csleft
        cmp.b #"2",row
        bgt csup
        cmp.b #"3",row+1
        bgt csup
        bra getasc                ;cursor is upper left

csleft: cmp.b #"0",col+1          ;set cursor
        bne.s m5
        move.b #"9"+1,col+1
        sub.b #1,col
m5:     sub.b #1,col+1
        sub.l #1,buffptr          ;left buffer pointer
```

```
        jsr cursor
        bra getasc

csup:   cmp.b #"0",row+1          ;cursor to end of line
        bne.s m6
        move.b #"9"+1,row+1
        sub.b #1,row
m6:     sub.b #1,row+1
        move.b #"7",col
        move.b #"0",col+1
        sub.l #1,buffptr
        jsr cursor
        bra getasc

ascup:  cmp.b #"2",row            ;cursor up if possible
        bgt rowup
        cmp.b #"3",row+1
        bgt rowup
        bra getasc

rowup:  cmp.b #"0",row+1
        bne.s m8
        move.b #"9"+1,row+1
        sub.b #1,row
m8:     sub.b #1,row+1
        sub.l #$40,buffptr        ;buffer pointer upper line
        jsr cursor
        bra getasc

nocurs: cmp.b #$1b,key            ;escape key= end input
        beq ascend
        cmp.b #" ",key            ;mask key
        blt getasc
        cmp.b #"z",key
        bgt getasc

        cmp.b #"7",col            ;char left or start of line
        blt.s doright             ;print
        cmp.b #"0",col+1
        blt.s doright
        cmp.b #"3",row
        blt dodown
        cmp.b #"0",row+1
        blt dodown
        jsr printkey
        move.l buffptr,a0         ;store in buffer
        move.b key,(a0)
        bra asciiedit             ;cursor home

doright:cmp.b #"9",col+1          ;char left print
        bne.s m0
        move.b #"0"-1,col+1
        add.b #1,col
m0:     add.b #1,col+1
        jsr printkey
        move.l buffptr,a0
```

```
        move.b key,(a0)         ;store in buffer
        add.l #1,buffptr
        jsr cursor
        bra getasc

dodown: cmp.b #"9",row+1        ;print char at start of line
        bne.s m1
        move.b #"0"-1,row+1
        add.b #1,row
m1:     add.b #1,row+1
        move.b #"0",col
        move.b #"7",col+1
        jsr printkey
        move.l buffptr,a0
        move.b key,(a0)         ;store in buffer
        add.l #1,buffptr
        jsr cursor
        bra getasc

ascend: jsr crsroff             ;end the  ascii input
;       jsr dumphex             ;hex output PAL only
        bra main

hexedit: move.b #'h',display    ;added NTSC
         jsr dumphex            ;added NTSC
        move.b #"0",row         ;hex input in buffer
        move.b #"6",row+1       ;similar to ascii input
        move.b #"0",col         ;except:   cursor in 2 steps
        move.b #"7",col+1       ;          to enter in bytes
        move.l buffer,buffptr
        jsr crsron
        jsr cursor
gethex: jsr getkey
        cmp.b #$9b,key
        bne noxcurs
        jsr getkey
        cmp.b #$44,key
        beq hexleft
        cmp.b #$43,key
        beq hexright
        cmp.b #$41,key
        beq hexup
        cmp.b #$42,key
        beq hexdown
        bra gethex

hexright:
        cmp.b #"6",col
        blt.s xcsright
        cmp.b #"9",col+1
        blt.s xcsright
        cmp.b #"2",row
        blt xcsdown
        cmp.b #"1",row+1
        blt xcsdown
        bra gethex
```

```
xcsright:
        cmp.b #"9",col+1
        bne.s n3
        move.b #"1"-2,col+1
        add.b #1,col
n3:     add.b #2,col+1
        add.l #1,buffptr
        jsr cursor
        bra gethex

xcsdown:cmp.b #"9",row+1
        bne.s n2
        move.b #"0"-1,row+1
        add.b #1,row
n2:     add.b #1,row+1
        move.b #"0",col
        move.b #"7",col+1
        add.l #1,buffptr
        jsr cursor
        bra gethex

hexdown:cmp.b #"2",row
        blt rowxdown
        cmp.b #"1",row+1
        blt rowxdown
        bra gethex

rowxdown:
        cmp.b #"9",row+1
        bne.s n4
        move.b #"0"-1,row+1
        add.b #1,row
n4:     add.b #1,row+1
        add.l #$20,buffptr
        jsr cursor
        bra gethex

hexleft:cmp.b #"0",col
        bgt.s xcsleft
        cmp.b #"7",col+1
        bgt.s xcsleft
        cmp.b #"0",row
        bgt xcsup
        cmp.b #"6",row+1
        bgt xcsup
        bra gethex

xcsleft:cmp.b #"1",col+1
        bne.s n5
        move.b #"9"+2,col+1
        sub.b #1,col
n5:     sub.b #2,col+1
        sub.l #1,buffptr
        jsr cursor
        bra gethex
```

```
xcsup:  cmp.b #"0",row+1
        bne.s n6
        move.b #"9"+1,row+1
        sub.b #1,row
n6:     sub.b #1,row+1
        move.b #"6",col
        move.b #"9",col+1
        sub.l #1,buffptr
        jsr cursor
        bra gethex

hexup:  cmp.b #"0",row
        bgt xrowup
        cmp.b #"6",row+1
        bgt xrowup
        bra gethex

xrowup: cmp.b #"0",row+1
        bne.s n8
        move.b #"9"+1,row+1
        sub.b #1,row
n8:     sub.b #1,row+1
        sub.l #$20,buffptr
        jsr cursor
        bra gethex


noxcurs:cmp.b #$1b,key
        beq hexend
        cmp.b #"0",key
        blt gethex
        cmp.b #"f",key
        bgt gethex
        cmp.b #"9",key
        ble.s ok0
        cmp.b #"a",key
        bge.s ok0
        bra gethex
ok0:    jsr printkey
ok2:    move.l wdhd,d1
        move.l #key2,d2
        moveq #1,d3
        jsr Read(a6)
        cmp.b #"0",key2
        blt.s ok2
        cmp.b #"f",key2
        bgt.s ok2
        cmp.b #"9",key2
        ble.s ok1
        cmp.b #"a",key2
        blt.s ok2

ok1:    move.b key,d0           ;convert key and key2 into byte
        cmp.b #"9",d0
        bgt.s ok3
```

```
         sub.b #"0"-"a"+10,d0
ok3:     sub.b #"a"-10,d0
         lsl.b #4,d0
         move.b d0,byte
         move.b key2,d0
         cmp.b #"9",d0
         bgt.s ok4
         sub.b #"0"-"a"+10,d0
ok4:     sub.b #"a"-10,d0
         or.b d0,byte

         cmp.b #"6",col
         blt.s doxright
         cmp.b #"9",col+1
         blt.s doxright
         cmp.b #"2",row
         blt doxdown
         cmp.b #"1",row+1
         blt doxdown
         move.l wdhd,d1
         move.l #key2,d2
         moveq #1,d3
         jsr Write(a6)
         move.l buffptr,a0
         move.b byte,(a0)
         bra hexedit

doxright:
         cmp.b #"9",col+1
         bne.s n0
         move.b #"1"-2,col+1
         add.b #1,col
n0:      add.b #2,col+1
         move.l wdhd,d1
         move.l #key2,d2
         moveq #1,d3
         jsr Write(a6)
         move.l buffptr,a0
         move.b byte,(a0)
         add.l #1,buffptr
         jsr cursor
         bra gethex

doxdown:cmp.b #"9",row+1
         bne.s n1
         move.b #"0"-1,row+1
         add.b #1,row
n1:      add.b #1,row+1
         move.b #"0",col
         move.b #"7",col+1
         move.l wdhd,d1
         move.l #key2,d2
         moveq #1,d3
         jsr Write(a6)
         move.l buffptr,a0
         move.b byte,(a0)
```

```
        add.l #1,buffptr
        jsr cursor
        bra gethex

hexend: jsr crsroff              ;end hex input
;       jsr dumpasc              ;ascii output  PAL only
        bra main

;       TEXT, VARIABLES AND TABLES

title:  dc.b "raw:0/0/640/200/"  ;PAL= "raw:0/0/640/256/"
        dc.b "      DISK-MONITOR VERSION 1.0    "
        dc.b "      INSERT DISK TO EXAMINE IN DF"
drive:  dc.b 0,":    ",0
top:    dc.b $0a

    ;   invers                1.char        normal  rest

    dc.b "            "
    dc.b " ",$9b,"0;31;43",$6d,"Esc",$9b,"0;31;40",$6d,"ape"
    dc.b " ",$9b,"0;31;43",$6d,"#",$9b,"0;31;40",$6d," Block"
    dc.b " ",$9b,"0;31;43",$6d,"$",$9b,"0;31;40",$6d," Block"
    dc.b " ",$9b,"0;31;43",$6d,"+",$9b,"0;31;40",$6d," Up"
    dc.b " ",$9b,"0;31;43",$6d,"-",$9b,"0;31;40",$6d," Down"
    dc.b " ",$9b,"0;31;43",$6d,"R",$9b,"0;31;40",$6d,"ead"
    dc.b " ",$9b,"0;31;43",$6d,"W",$9b,"0;31;40",$6d,"rite"
    dc.b " ",$9b,"0;31;43",$6d,"C",$9b,"0;31;40",$6d,"hecksum"
    dc.b " ",$9b,"0;31;43",$6d,"A",$9b,"0;31;40",$6d,"scii"
    dc.b " ",$9b,"0;31;43",$6d,"H",$9b,"0;31;40",$6d,"ex"
    dc.b $0a,$0a
    dc.b " Block #     $      Buffer #     $      Checksum $"
    dc.b $0a,"----------------------------------------------------"
    dc.b "--------------------------"
topend:
toplen=topend-top

        ; BLOCKTYP

boot:   dc.b "BOOTBLOCK "
root:   dc.b "ROOTBLOCK "
flist:  dc.b "FILELIST  "
fhead:  dc.b "FILEHEADER"
dat:    dc.b "DATABLOCK "
udir:   dc.b "USERDIR   "
unkn:   dc.b "----------"

        ; DISK Error messages

nderr:  dc.b $9b,"43",$6d,"NO DISK IN DRIVE !",$9b,"40",$6d
nderrend:
ndlen=nderrend-nderr
rderr:  dc.b $9b,"43",$6d,"   READ-ERROR !    ",$9b,"40",$6d
rdend:
rdlen=rdend-rderr
wrerr:  dc.b $9b,"43",$6d,"   WRITE-ERROR !   ",$9b,"40",$6d
wrend:
```

```
wrlen=wrend-wrerr
pterr:  dc.b $9b,"43",$6d,"WRITE-PROTECTION !",$9b,"40",$6d
ptend:
ptlen=ptend-pterr
clear:  dc.b "    NO ERROR !    "
clrend:
clrlen=clrend-clear


 align   ;even
crsrstatus:
        dc.w 0
adr:    dc.w 0
mytext:   dc.b "0000"
key:    dc.b 0
key2:   dc.b 0
display dc.b 0
byte:   dc.b 0
 align ;even
linebuf:blk.b 64,0      ;buffer for conversions

        ;sequence for cursor positioning

pos:    dc.b $9b
row:    dc.b "00",$3b
col:    dc.b "00",$48

        ;s.o. for address number dec and hex

adrpos: dc.b $9b,"04",$3b,"29",$48
adrpos2:dc.b $9b,"04",$3b,"35",$48

        ;sequence for cursor on/off

con:    dc.b $9b,$20,$70
coff:   dc.b $9b,$30,$20,$70

dosname:dc.b "dos.library",0
trkdisk:dc.b "trackdisk.device",0
device: dc.b 0

 align   ;even
dosbase:dc.l 0
wdhd:   dc.l 0          ;window handle

block:  dc.w 880        ;startblock
offset: dc.l 0          ;offset for read/write =512*block
buffptr:dc.l 0          ;buffer pointer
buffer: dc.l 0          ;buffer start

diskport:
        dc.l 0          ;0
        dc.l 0          ;4
        dc.w $0400      ;8
        dc.l 0          ;10
        dc.b 0          ;14
```

```
        dc.b 31         ;15
        dc.l 0          ;16     task adr. here
LH1:    dc.l LH2        ;20
LH2:    dc.l 0          ;24
        dc.l LH1        ;28
        dc.b 0          ;32
        dc.b 0          ;33

diskioreq:
        dc.l 0          ;0
        dc.l 0          ;4
        dc.b 5          ;8
        dc.b 0          ;9
        dc.l 0          ;10
        dc.l diskport   ;14
        dc.w 48         ;18
        dc.l 0          ;20
        dc.l 0          ;24
        dc.w 0          ;28 IO_CMD
        dc.w 0          ;30
        dc.l 0          ;32 IO_ERROR
        dc.l 0          ;36 IO_LENGTH
        dc.l 0          ;40 IO_DATA
        dc.l 0          ;44 IO_OFFSET
        dc.l 0
        dc.l 0
;The following is for NTSC versions only, to clear 8 hex lines
clrhex: dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
        dc.b $0a,"                                              "
        dc.b "                          "
clrhexend:
clrhexlen = clrhexend-clrhex
 end
```

Appendix B
The Drive Accelerator

Most users have been annoyed by the slow access time of the Amiga disk drives. This is caused partly by a complex and therefore slow system, and partly by the complicated routines of the Trackdisk device.

The main reason for slow loading is in the basic principle of the Amiga operating system: everything should be easy to expand.

To satisfy this principle, the Load command is first sent from DOS to the Filesystem. Here the command must first be recognized and then transmitted to the Trackdisk device.

The desired tracks are read from disk into memory and the data passed to the Filesystem with the help of the blitter. Some of these are temporarily stored and the data is passed to DOS which pushes it to the final location in memory. Because of the frequent passing of commands and data to other parts of the system, much time is lost. Without reorganizing the system completely, there is no way to speed this process.

Another possibility, however, is to accelerate the Trackdisk device since it controls the reading of a track in a very complicated manner. The accelerator presented here uses this method.

To use the disk accelerator described in this book, the present system and its weaknesses must first be explained.

The Filesystem sends a read command to the Trackdisk device. The Filesystem decides what should be read. As soon as the command (read disk block into a certain buffer) is passed to the Trackdisk device (more exactly the Trackdisk task), it starts to work.

The command is tested for its legality and at the same time a jump is performed to the proper routine for further processing. Now a test is made to determine if the track, whose block is needed, is already in memory. If this is the case, the block is decoded by the blitter (from MFM format to normal code) and copied into the buffer desired by the Filesystem.

If the track was not already in memory, it must be loaded. The loading of a track has been implemented in a very complicated and not very elegant manner.

A track contains, including track gaps, about $3100 bytes, depending on the drive in use. To insure that the Trackdisk device has read the entire track, $397C bytes are loaded. For some unknown reason there is no wait for the sync mark during reading. The reading is unsynchronized. For this reason the sync mark must be found "by hand", which is not simple. First it must discover if the first bit read was a track or data bit. After finding the mark, the data, with the sync marking at the front, are copied properly and checked for read errors. Only after the data has been verified is the block decoded and moved to the buffer. Then the Filesystem continues its work.

Using the accelerator

The first problem when modifying the operating system is how to insert a user routine into the system. The easiest possibility is to set the return jump address of the Trackdisk task to the user task, which is similar to it, including the load routines. After this has been done, a branch occurs not to ROM, but to the user routine, every time an operation is performed on the disk.

In the user task, just as in the system, the commands are tested for legality. If it is not a command to read a block, a jump is performed to the operating system. This must occur through direct addresses. This is also why the accelerator only runs with Kickstart Version 33.192 (of the Amiga 500 and 2000). In other versions, the absolute addresses are different.

If the display blinks after starting, the accelerator is working. If not, there is either a wrong Kickstart version or not enough memory could be made available.

When the command to read is passed, the program runs in the user routines which are completely different from the operating system.

To prevent reading substantially more data than one track length, the header of the first block found is read. During the read a wait occurs for the sync mark. On the basis of the data read, it can be recognized where the track gap is located and how many bytes can be found in front and behind it. It takes little time to find this information and the following block can be read directly.

After the number of bytes to the gap has been discovered, they can be read after finding the sync mark. After the gap there must be another wait for the "Sync", after which the rest of the data is read. It is therefore no longer necessary to read $397CK, only the actual track length in addition to the block read for orientation. This of course increases the speed. In addition, the data does not have to be corrected by copying as in the operating system, because a wait occurred for the sync mark.

The data is read from the disk directly to memory through DMA (Direct Memory Access), without the use of the processor. You can therefore have the processor decode the data and check for errors during the same time the data is read from disk. With this system the assignments normally performed sequentially are performed simultaneously.

When the Filesystem requests a block whose track is already in the memory of the Trackdisk device, it doesn't have to be decoded first by the operating system. It can be copied to the desired location immediately by the blitter (the decoding was done during the reading).

To achieve an even more acceleration, the speed at which the head is moved across the disk is increased. Also the wait time for the device, after positioning the head to the desired location, is set to almost zero. This is very noticeable in loading times.

Altogether the routines of the Trackdisk device which are responsible for the reading of a track, have been accelerated to the maximum.

As mentioned earlier, most of the time for reading a file is not used by the Trackdisk device, but the complicated system which, for reasons of compatibility, cannot be changed. The accelerator described here is an improvement, and is about as much as is possible through the enhancement of the Trackdisk devices.

The source listing for the accelerator follows. It is also contained on the optional disk for this book. This program was assembled with the AssemPro asembler from Abacus.

```
;Listing of floppy accelerator program
;speeder.s
;from the Abacus book
;Amiga Disk Drives Inside and Out
;Assembled using the AssemPro Assembler

DeviceList      EQU 350
TrackTask       EQU 302
TrackPort       EQU 36
SPReg           EQU 54
ReplyAddress    EQU 70
IDNestCnt       EQU 294
PortStatus      EQU 34
CMD_READ        EQU 2
FindName        EQU -276
Wait            EQU -318
AllocMem        EQU -198
FreeMem         EQU -210

Req1            EQU $01                    ;MEMF_PUBLIC
Req2            EQU $03
TrackSize       EQU $1604                  ;Number of Bytes in one
Track

ReadError       EQU 21
NoDisk          EQU 29
NoSync          EQU 21

                lea $fc0000,a0
                cmp.l #$2033332E,$1c(a0)
                bne \DError4
                cmp.l #$31393220,$20(a0)
                bne \DError4
                move.l $4,a6
                bsr Disable
                move.l #Ende-Start1,d0
                move.l #Req1,d1
                jsr AllocMem(a6)
                move.l d0,a1
                move.l a1,a4               ;Save memory address
                move.l d0,d4
                beq \DError1
                lea TrackName(pc),a1
                lea DeviceList(a6),a0
                jsr FindName(a6)
                tst.l d0
                beq \DError3
                move.l d0,a5

                lea MyTask(pc),a0
                lea Start1(pc),a1
                suba.l a1,a0
                adda.l a0,a4               ;Address of  MyTask
                clr.l d3
\13:            move.l TrackPort(a5,d3),d0
                beq \15
```

```
                move.l d0,a3
                move.l #TrackSize,d0
                move.l #Req2,d1
                jsr AllocMem(a6)
                lea TrackMemory1(pc),a0
                lea (a0,d3),a0          ;Track Memory for Disk
                move.l d0,(a0)
                beq \DError2
\l1:            btst #0,PortStatus(a3)  ;wait, until Task in Wait
                bne \l1
                move.l #1800,$2c(a3)    ;accelerate Step motor
                move.l #1,$30(a3)       ;no wait after Posi.
                lea TrackTask+SPReg(a3),a2
                move.l (a2),a1
                move.l a4,ReplyAddress(a1)
\l5:            addq.l #4,d3
                cmpi.w #16,d3
                bcs \l3

                move.l #Ende-Start1,d0
                lea Start1(pc),a0
                move.l d4,a1
\l4:            move.b (a0)+,(a1)+      ;Copy data
                subq.l #1,d0
                bne \l4
                bsr blink
\DError1:
                bsr Enable
\DError4:       clr.l d0
                rts
\DError2:       subq.l #4,d3
                bcs \DError3
                lea TrackMemory1(pc),a0
                lea (a0,d3),a0
                move.l (a0),a1
                move.l #TrackSize,d0
                jsr FreeMem(a6)
                bra \DError2
\DError3:       move.l #Ende-Start1,d0
                move.l a4,a1
                jsr FreeMem(a6)
                bra \DError1

Blink:          move.l D0,-(a7)
                move.l #$20000,d0
\l1:            move.w d0,$dff180
                sub.l #1,d0
                bne \l1
                move.l (a7)+,D0
                rts

Start1:
Disable:        move.w #$4000,$dff09a
                move.l a6,-(a7)
                move.l $4,a6
                add.b #1,IDNestCnt(a6)
```

```
                move.l (a7)+,a6
                rts
Enable:         move.l a6,-(a7)
                move.l $4,a6
                sub.b #1,IDNestCnt(a6)
                bge \l1
                move.w #$c000,$dff09a
\l1:            move.l (a7)+,a6
                rts

TheTask:        MOVEA.L 8(A7),A6
                MOVEA.L 4(A7),A3
                LEA $12E(A3),A0
                MOVE.L A0,$10(A3)
                JSR $FE9960
LFEAE64:        BSR.S LFEAE7A
                MOVE.L #$300,D0
                MOVE.L A6,-(A7)
                MOVEA.L $34(A6),A6
                JSR Wait(A6)
MyTask:         MOVEA.L (A7)+,A6
                BRA.S LFEAE64
LFEAE7A:        BSET #0,$22(A3)
                BNE NoMessage
LFEAE84:        MOVEA.L A3,A0
                MOVE.L A6,-(A7)
                MOVEA.L $34(A6),A6
                JSR -$174(A6)
                MOVEA.L (A7)+,A6
                TST.L D0
                BEQ LFEAF3E
                MOVEA.L D0,A2
                BCLR #3,$40(A3)
                BEQ LFEAF1E
                MOVEA.L $52(A3),A0
                BCLR #0,2(A0)
                BEQ LFEAEBA
                MOVE.L A0,$4E(A3)
                JSR $FEA958
LFEAEBA:        MOVEA.L $52(A3),A0
                MOVEQ #-1,D0
                MOVE.W D0,0(A0)
                MOVE.W D0,$4C(A3)
                MOVEQ #0,D0
                JSR $FEA462
                MOVEA.L A6,A0
                MOVEA.L $34(A0),A6
                ADDQ.B #1,$127(A6)
                MOVEA.L A0,A6
                TST.W $24(A3)
                BNE LFEAF12
                MOVEQ #0,D0
                MOVE.B $43(A3),D0
                MOVE.L A6,-(A7)
                MOVEA.L $3C(A6),A6
                JSR -$C(A6)
```

```
                MOVEA.L (A7)+,A6
                LEA $24(A6),A0
                MOVEQ #0,D0
                MOVE.B $43(A3),D0
                LSL.L #2,D0
                ADDA.L D0,A0
                CLR.L (A0)
                SUBA.L A1,A1
                MOVE.L A6,-(A7)
                MOVEA.L $34(A6),A6
                JSR -$120(A6)
                MOVEA.L (A7)+,A6
LFEAF12:        MOVE.L A6,-(A7)
                MOVEA.L $34(A6),A6
                JSR -$8A(A6)
                MOVEA.L (A7)+,A6
LFEAF1E:        MOVEA.L A2,A1
                LEA $86(A3),A0
                CMPA.L A0,A2
                BNE.S LFEAF30
                JSR $FE9960
                BRA LFEAE84
LFEAF30:        BSET #1,$22(A3)
                bsr Stepper1
                BRA LFEAE84
LFEAF3E:        BCLR #1,$22(A3)
                BCLR #0,$22(A3)
NoMessage:      RTS

Stepper1:       MOVE.L A2,-(A7)
                MOVEA.L A1,A2
                ANDI.B #-6,$40(A3)
                jsr $FE998C
                MOVEA.L A2,A1
                MOVE.W $1C(A2),D0
                cmp.b #CMD_READ,d0
                beq Stepper2
                BTST #$F,D0
                BEQ.S LFEA052
                BSET #2,$40(A3)
                MOVE.L $126(A3),D1
                CMP.L $30(A2),D1
                BLS.S LFEA052
                MOVE.B #$1D,$1F(A2)
                JSR $FEA1B0
                BRA.S LFEA066
LFEA052:        MOVEQ #0,D1
                MOVE.B D0,D1
                LSL.W #2,D1
                LEA $FEA300,A0
                MOVEA.L 0(A0,D1.W),A0
                JSR (A0)
LoadEnde:
                JSR $FE998C
LFEA066:        MOVEA.L (A7)+,A2
                RTS
```

```
Stepper2: ;      bsr Blink
                 bsr Stepper3
                 bra LoadEnde

Stepper3:
                 MOVEM.L A2-A4,-(A7)
                 MOVEA.L $18(A1),A3
                 MOVEA.L A1,A2
                 MOVE.L A2,$44(A3)
                 MOVE.L #0,$20(A2)
                 MOVE.L $28(A2),$56(A3)
                 MOVE.L $2C(A2),D0
                 JSR $FEA182
                 TST.L D0
                 BMI LFEA92A
                 MOVE.W D0,$4A(A3)
                 MOVE.B D1,$49(A3)
                 MOVE.L $2C(A2),D0
                 ADD.L $24(A2),D0
                 JSR $FEA182
                 TST.L D0
                 BMI LFEA92A
                 BTST #2,$40(A3)
                 BEQ.S LFEA78E
                 MOVE.L $34(A2),$5A(A3)
                 BEQ.S LFEA78E
                 BSET #0,$40(A3)
LFEA78E:         BTST #1,$40(A3)
                 BEQ.S LFEA7A0
                 MOVE.B #$1D,$1F(A2)
                 BRA LFEA920
LFEA7A0:         MOVE.W $4A(A3),D0
                 JSR $FEA93C
                 MOVE.L A0,$4E(A3)
                 BNE.S LFEA804
                 jsr $FEA952
                 MOVEA.L A0,A2
                 MOVE.L A2,$4E(A3)
LFEA7B8:         BTST #0,2(A2)              ;Track in buffer
                 BEQ.S LFEA7D4              ;No, read one
                 JSR $FEA958
                 TST.L D0
                 BEQ.S LFEA7D4
                 MOVEA.L $44(A3),A1
                 MOVE.B D0,$1F(A1)
                 BRA LFEA920
LFEA7D4:         MOVE.W $4A(A3),0(A2)
                 BCLR #0,2(A2)
                 CLR.B $42(A3)
                 bsr read1                ;read track into buffer
                 MOVE.B 3(A2),D0
                 CMPI.B #$B,D0
                 BCS.S LFEA804
                 MOVE.W #-1,0(A2)
                 MOVEA.L $44(A3),A1
```

```
                MOVE.B D0,$1F(A1)
                BRA LFEA920
LFEA804:        MOVEA.L $44(A3),A2
                MOVE.W $1C(A2),D0
                MOVEA.L $4E(A3),A0
                CMPI.B #3,D0
                BNE LFEA890
                BSET #0,2(A0)
                MOVEQ #0,D0
                MOVE.B $49(A3),D0
                SUB.B 3(A0),D0
                BPL.S LFEA82E
                ADDI.B #$B,D0
LFEA82E:        MULU #$440,D0
                LEA $680(A0),A4
                ADDA.L D0,A4
                BTST #0,$40(A3)
                BEQ.S LFEA866
                MOVEA.L $5A(A3),A0
                MOVE.L #$10,D0
                LEA $10(A4),A1
                JSR $FEAB4A
                LEA 8(A4),A0
                MOVE.W #$28,D1
                JSR $FEADA4
                LEA $30(A4),A0
                JSR $FEAD46
LFEA866:        MOVEA.L $56(A3),A0
                MOVE.L #$200,D0
                LEA $40(A4),A1
                JSR $FEAB4A
                LEA $40(A4),A0
                MOVE.W #$400,D1
                JSR $FEADA4
                LEA $38(A4),A0
                JSR $FEAD46
                BRA LFEA8D6
LFEA890:        MOVEQ #0,D0
                BTST #0,$40(A3)          ;SecLabel set?
                BEQ.S LFEA8C4            ;no
                MOVE.B $49(A3),D0        ;Sector number to D0
                SUB.B 3(A0),D0
                BPL.S LFEA8A0
                ADDI.B #$B,D0
LFEA8A0:        MULU #$440,D0
                LEA $680(A0),A4
                ADDA.L D0,A4
                LEA $10(A4),A1
                MOVEA.L $5A(A3),A0
                MOVE.L #$10,D0
                JSR $FEACB2
LFEA8C4:
                clr.l d0
                MOVE.B $49(A3),D0           ;Sector number to D0
                MOVEA.L $56(A3),A1          ;destination address
                lea TrackMemory1(pc),a0
```

```
                clr.l d1
                move.b $43(a3),d1
                lsl.w #2,d1
                adda.l d1,a0
                movea.l (a0),a0
                mulu #$200,d0
                adda.l d0,a0
                move.w #$200,d0
                bsr CopyBlock
LFEA8D6:        MOVE.L #$200,D1
                ADD.L D1,$56(A3)
                MOVE.L $20(A2),D0
                ADD.L D1,D0
                MOVE.L D0,$20(A2)
                BTST #0,$40(A3)
                BEQ.S LFEA8FA
                ADDI.L #$10,$5A(A3)
LFEA8FA:        CMP.L $24(A2),D0
                BCC.S LFEA920
                MOVEA.L $4E(A3),A2
                ADDQ.B #1,$49(A3)
                CMPI.B #$B,$49(A3)
                BLT LFEA804
                MOVE.B #0,$49(A3)
                ADDQ.W #1,$4A(A3)
                BRA LFEA7B8
LFEA920:        MOVEA.L $44(A3),A1
                JSR $FEA1B0
                BRA.S LFEA936
LFEA92A:        MOVEA.L $44(A3),A1
                MOVE.B #-4,$1F(A1)
                BRA.S LFEA920
LFEA936:        MOVEM.L (A7)+,A2-A4
                RTS
readl:          MOVEM.L A2,-(A7)
                MOVEA.L $4E(A3),A2
                MOVEQ #1,D0
                jsr $FEA462                 ;Motor on
LFEA9AC:        MOVEQ #0,D0
                MOVE.W $4A(A3),D0
                jsr $FEA3DA                 ;Head Posi.
LFEA9B6:        LEA 1664(A2),A0
                lea TrackMemory1(pc),a1
                clr.l d0
                move.b $43(a3),d0
                lsl.w #2,d0
                adda.l d0,a1
                move.l (a1),a1
                bsr trackread1
                move.w FirstBlock(pc),D0
                MOVE.B d0,3(A2)
                lea ErrorFlag(pc),a0
                tst.w (a0)
                beq \Ende                   ;No Error
\Error:         MOVE.B 1(a0),3(A2)          ;Store error
                ADDQ.B #1,$42(A3)
```

```
              MOVE.B $42(A3),D0
              CMP.B $34(A3),D0
              BGT.S \Ende             ;End too many errors
              ANDI.B #3,D0
              BNE.S LFEA9B6
              MOVE.W #-1,$4C(A3)
              BRA.S LFEA9AC
\Ende:        MOVEM.L (A7)+,A2
              RTS

;Track read and decoder
;>= A1 = Pointer to buffer for decoded data
;>= A0 = Pointer to buffer for coded data

Trackread1:
              MOVEM.L D2-D4/a4-a5,-(A7)
              move.l a0,a5
              move.l a1,a4
              lea ErrorFlag(pc),a1
              clr.w (a1)
              lea DecodeNum(pc),a1
              move.w #$080,(a1)     ;Number long words to  decode
              lea $40(a5),a0
              lea DecodeAdr(pc),a1
              move.l a0,(a1)          ;Data area for 1. Blocks
              adda.l #$400,a0
              lea FTestAdr(pc),a1
              move.l a0,(a1)          ;Address of next Block
              jsr $FEADDC             ;Check drive
              MOVE.B $41(A3),$BFD100
              BTST #2,$BFE001         ;Disk in drive
              BNE.S \FL3              ;Ok
              lea ErrorFlag(pc),a1
              move.w #NoDisk,(a1)     ;No disk in drive
              BRA   \FL5              ;Ende

\FL3:         bsr Disable
              move.l a6,-(a7)
              MOVEA.L A5,A6           ;Track buffer
              move.l #$aaaaaaaa,(a6)+
              move.w #$4489,(a6)+     ;store first Sync
              bsr search
              tst.l d0
              bpl \FL8
              lea ErrorFlag(pc),a1
              move.w #NoSync,(a1)     ;No Sync
              bra \FL9                ;End
\FL8:         bsr FErase              ;Prepare track buffer
              clr.l d2
              move.w BytesBefGap(pc),d2 ;Num. of Bytes before Gap
              tst.l d2
              beq \FL1                ;No Bytes before Gap
              lea BlockAdr(pc),a1
              clr.w (a1)              ;Offset im Block
              bsr Numread1            ;Bytes read
              clr.l d0
```

```
                move.w BytesBefGap(pc),d0
                move.l a5,a6
                adda.l d0,a6            ;Pointer to next buffer
                move.l #$aaaaaaaa,(a6)+
                move.w #$4489,(a6)+     ;store first sync
\FL1:           move.w BytesAftGap(pc),d2
                tst.l d2
                beq \FL2
                lea BlockAdr(pc),a1
                clr.w (a1)
                bsr Numread1
                bsr lastoneblock
\FL2:           move.l #$aaaaaaaa,$2ec0(a5) ;Creat gap after data
                BTST #2,$BFE001         ;Disk in drive?
                bne   \FL9              ;Ok, Disk in Drive
                lea ErrorFlag(pc),a1
                move.w #NoDisk,(a1)
\FL9:           move.l (a7)+,a6
\FL5:           bsr Enable
                jsr $FEAE42             ;drop drive
                MOVEM.L (A7)+,D2-D4/a4-a5
                RTS

;Prepare track buffer (clrear block start)
;>= A5 = Pointer to track buffer

FErase:         move.l a5,a0
                move.w #10,d1
                clr.l d0
\L1:            move.l d0,$440(a0)
                adda.l #$440,a0
                dbf d1,\L1
                lea BlockReport1(pc),a0
                move.w #10,d1
\L2:            clr.w (a0)+
                dbf d1,\L2
                rts

;Read set number of bytes
;>= A6 = Pointer to destination
;>= D2 = Number of bytes to read

Numread1:
                bsr install
                MOVE.W D2,D0
                LSR.W #1,D0
                ORI.W #$8000,D0
                add.w #1,d0
                MOVE.W D0,36(A1)
                MOVE.W D0,36(A1)
                bsr decode
                LEA $DFF000,A1
                MOVE.W #$4000,$24(A1)
                rts

;Prepare to read
```

```
;>= A6 Pointer to track buffer

install:        LEA $DFF000,A1
                move.w #$4000,$24(a1)     ;set Disk-Len back
                move.w #$8400,$9e(a1)     ;switch on Disk Sync
                move.w #$4489,$7e(a1)     ;SYNC-Mark
                MOVE.L A6,$20(A1)         ;pass buffer
                move.w #$0002,$dff09c
                rts

;Code long word and enter into buffer
;>= D0 = Long word
;>= A0 = Pointer to buffer

CodeLWort:      MOVEM.L D2-D3,-(A7)
                MOVE.L D0,D3
                LSR.L #1,D0
                BSR \CH1
                MOVE.L D3,D0
                BSR \CH1
                BSR Randsetone
                MOVEM.L (A7)+,D2-D3
                RTS
\CH1:           ANDI.L #$55555555,D0
                MOVE.L D0,D2
                EORI.L #$55555555,D2
                MOVE.L D2,D1
                LSL.L #1,D2
                LSR.L #1,D1
                BSET #$1F,D1
                AND.L D2,D1
                OR.L D1,D0
                BTST #0,-1(A0)
                BEQ.S \CH2
                BCLR #$1F,D0
\CH2:           MOVE.L D0,(A0)+
                RTS
;set border

Randsetone:     MOVE.B (A0),D0
                BTST #0,-1(A0)
                BNE.S \CH4
                BTST #6,D0
                BNE.S \CH6
                BSET #7,D0
                BRA.S \CH5
\CH4:           BCLR #7,D0
\CH5:           MOVE.B D0,(A0)
\CH6:           RTS

;determine checksum
;>= D1 = Number of Bytes (must be divisible by 4)
;>= A0 = Pointer to buffer
;=> D0 = Check sum

CheckSum:       MOVE.L D2,-(A7)
```

```
                LSR.W #2,D1
                SUBQ.W #1,D1
                MOVEQ #0,D0
\PS1:           MOVE.L (A0)+,D2
                EOR.L D2,D0
                DBRA D1,\PS1
                ANDI.L #$55555555,D0
                MOVE.L (A7)+,D2
                RTS

;Decode block header
;>= A0 is pointer to header
;=> D0 = Header

Header:         move.l (a0)+,D0
                move.l (a0)+,D1
                andi.l #$55555555,d0
                andi.l #$55555555,d1
                lsl.l #1,D0
                or.l D1,D0
                rts

;find first block
;=> A6 = Pointer to track buffer
;=> D0 = Null: Block found
;=> BytesBefGap = Number of Bytes before the Gap
;=> BytesAftGap = Number of Bytes after the Gap

search:
                movem.l d2-d4/a2,-(a7)
                move.w #11,d2           ;Number of errors permitted
\SU1:           bsr install
                move.w #$8024,d0          ;$24 Words read
                MOVE.W D0,$dff024
                MOVE.W D0,$dff024
                bsr Blockready            ;wait for ready Block
                tst.l d0                  ;Error, then D0 = -1
                bmi \SUError

                lea 8(a5),a0              ;Pointer to Blockheader
                moveq #$28,d1             ;number of long words
                bsr CheckSum              ;Sum for Header
                move.l d0,d3              ;Sum stored
                lea 48(a5),a0             ;*Sum
                bsr Header                ;get sum from Header
                cmp.l d0,d3               ;compare sums
                bne \SUNeu
                lea 8(a5),a0
                bsr Header                ;Header decode
                move.w d0,d3              ;Header to D3
                lsr.w #8,d3
                andi.w #$00ff,d3          ;isolate sector number
                addi.w #1,d3              ;incr. sector number
                cmp.w #$000a,d3           ;Nummer > 10?
                bls \SU2                  ;No, OK
                clr.w d3                  ;Number = 0
```

```
\SU2:           lea SectNum(pc),a2
                move.w d3,(a2)           ;Store number
                lea FirstBlock(pc),a2
                move.w d3,(a2)           ;Number of first block

                move.w d0,d3             ;Header
                andi.w #$ff,d3           ;Sectors to gap
                cmp.b #$0c,d3            ;Header OK?
                bcs.s \SUok
\SUNeu:         dbf d2,\SU1
                bra \SUError
\SUok:

                sub.w #1,d3              ;Num. of blocks to gap
                move.w d3,d2
                move.w #$000b,d4
                sub.b d2,d4              ;Num. of blocks after gap
                mulu #$440,d3            ;Num. of bytes to gap
                mulu #$440,d4            ;Num. of bytes to gap
                clr.l d0
                lea BytesBefGap(pc),a2
                move.w d3,(a2)
                lea BytesAftGap(pc),a2
                move.w d4,(a2)
                lea SectBL(pc),a2
                move.w #$0b,(a2)       ;Sectors before gap to load
                bra \SUEnd
\SUError:       move.l #-1,d0
                lea ErrorFlag(pc),a2
                move.w #ReadError,(a2)
\SUEnd:         movem.l (a7)+,d2-d4/a2
                rts

Blockready:     clr.l d0                 ;Error-Flag cleared
                move.l #$20000,d1
                move.w #$0002,$dff09c    ;DiskInt cleared
\B1:            MOVE.W $DFF01E,D0
                BTST #1,D0
                bne.s \B2
                sub.l #1,d1
                bne \B1
                move.l #-1,d0            ;Error occoured
\B2:            RTS


;decode bytes, unitl block ead

decode:
                movem.l d2-d4/a2-a3,-(a7)
                clr.l d3
                move.l a3,d4             ;save drive-prot
                move.w BlockAdr(pc),d3   ;Offset in Block
                move.l FTestAdr(pc),a0   ;Address to test if
                                         ;Block already loaded
                move.l DecodeAdr(pc),a2   ;Address, decode is done
                move.w DecodeNum(pc),d2      ;Number for decoding
```

```
\DC1:           MOVE.W $DFF01E,D0
                BTST #1,D0                  ;Area read already
                bne  \DCEnd                 ;Yes, end
                tst.l (a0)                  ;TestAdr
                beq \DC1                    ;Wait, until block read

                movem.l a0-a1,-(a7)         ;save registern
                lea -$40(a2),a1             ;* Block Start
                move.l d4,a3                ;* Drive-Port
                bsr BlockCheck              ;Block check
                movem.l (a7)+,a0-a1         ;restore Register
                move.w SectNum(pc),d0
                mulu #$200,d0
                move.l a4,a1            ;Basic address for dest. data
                add.l d0,a1                 ;Address of the Blocks
\DC2:
                MOVE.W $DFF01E,D0
                BTST #1,D0
                bne.s \DCEnd                ;area already read

                move.l (a2),D0
                move.l $200(a2),D1
                adda.l #4,a2
                andi.l #$55555555,d0
                andi.l #$55555555,d1
                lsl.l #1,D0
                or.l D1,D0
                move.l d0,(a1,d3)           ;store long word
                addq.w #4,d3
                subq.w #1,D2                ;Decode number
                bne \DC2
                adda.l #$240,a2             ;incr. Address
                adda.l #$440,a0             ;TestAdr
                move.l #$080,D2             ;Decode number
                clr.w d3                    ;Offset to Null
                lea SectNum(pc),a3
                add.w #1,(a3)               ;incr. Sector number
                cmp.w #$0b,(a3)             ;Nummer > 10?
                bcs \DC3                    ;No, OK
                clr.w (a3)                  ;Number = 0
\DC3:           bra \DC1

\DCEnd:
                lea BlockAdr(pc),a3
                move.w d3,(a3)
                lea DecodeAdr(pc),a3
                move.l a2,(a3)
                lea FTestAdr(pc),a3
                move.l a0,(a3)
                lea DecodeNum(pc),a3
                move.w D2,(a3)

                movem.l (a7)+,d2-d4/a2-a3
                RTS
```

```
;decode last block

lastoneblock:
                movem.l d2-d3/a2,-(a7)

                move.w SectNum(pc),d0
                mulu #$200,d0
                move.l a4,a1            ;Basic addres for dest. data
                add.l d0,a1               ;Address of the blocks
                clr.l d3
                move.l DecodeAdr(pc),a2
                move.w DecodeNum(pc),d2
\LB1:           move.l (a2),D0
                move.l $200(a2),D1
                adda.l #4,a2
                andi.l #$55555555,d0
                andi.l #$55555555,d1
                lsl.l #1,D0
                or.l D1,D0
                move.l d0,(a1,d3)
                addq.w #$4,d3
                subq.w #1,D2               ;Decode number
                bne \LB1
                movem.l (a7)+,d2-d3/a2

                RTS

;test Block for Errors
;A1 = Pointer to BlockStart

BlockCheck:
                movem.l d2-d3/a2,-(a7)
                clr.l d3
                move.w SectNum(pc),d3
                lsl.w #1,d3                ;Sector number => Offset
                lea BlockReport1(pc),a0
                move.w (a0,d3),d0          ;get entry
                tst.w d0                   ;already tested?
                bne \CBEnd2                ;Yes, end

                lea 64(a1),a0
                move.w #$400,d1
                bsr CheckSum               ;Sum for Data block
                move.l d0,d2               ;save sum
                lea 56(a1),a0              ;Pointer to Data sum
                bsr Header                 ;Sum decoder
                cmp.l d0,d2
                bne \DataIsFalse

                lea 8(a1),a0
                bsr Header                 ;Header decode
                move.w d0,d2               ;store lowere word
                lsr.w #8,d2                ;Sector number to d2
                cmp.b SectNum+1(pc),d2         ;rright Sector
                bne \FalseoneSector
                swap d0                    ;Track number to D0
```

```
                cmp.b 77(a3),d0          ;right Track?
                bne \FalseoneTrack
                andi.l #$ff00,d0
                cmp.w #$ff00,d0
                bne \KeinDosTrack
                lea 8(a1),a0
                moveq #$28,d1            ;long word number
                bsr CheckSum             ;Sum for Header
                move.l d0,d2             ;save sum
                lea 48(a1),a0            ;*Sum
                bsr Header               ;get sum from Header
                cmp.l d0,d2              ;compare sum
                bne \Header1False
                move.w #$ffff,d0
\CBEnd1:        lea BlockReport1(pc),a0
                move.w d0,(a0,d3)
                btst #0,-1(a1)
                beq \CB1
                move.l #$2aaaaaaa,(a1)
                bra \CB2
\CB1:           move.l #$aaaaaaaa,(a1)
\CB2:           move.l #$44894489,4(a1)
                move.w #$ff00,d0         ;create new Header
                move.b 77(a3),d0
                swap d0
                move.b SectNum+1(pc),d0
                lsl.w #8,d0
                move.b SectBL+1(pc),d0
                lea 8(a1),a0
                bsr CodeLWort            ;store Header
                lea 8(a1),a0
                moveq #$28,d1            ;long word number
                bsr CheckSum             ;Sum for Header
                lea 48(a1),a0            ;*Sum
                bsr CodeLWort            ;Checksum stored
                lea SectBL(pc),a2
                subq.w #1,(a2)
\CBEnd2:        movem.l (a7)+,d2-d3/a2
                rts

\FalseoneSector: move.w #$0017,d0
                 bra \Flagsetone
\FalseoneTrack: move.w #$0017,d0
                 bra \Flagsetone
\KeinDosTrack:  move.w #$0017,d0
                 bra \Flagsetone
\Header1False:  move.w #$001b,d0
                 bra \Flagsetone
\DataIsFalse:   move.w #$0019,d0
\Flagsetone:    lea ErrorFlag(pc),a2
                move.w d0,(a2)
                bra \CBEnd1

;Data bock coded
;>= D0 = Length of source
;>= A0 = Pointer to Source
```

```
;>= A1 = Pointer to Dest.

CopyBlock:
                move.l a2,-(a7)
                move.l a0,a2
                LSL.W #2,D0
                ORI.W #8,D0
                lea $dff000,a0
                bsr BlitWait
                bsr BlitterCode
                move.l (a7)+,a2
                RTS

;A0 = $dff000
;D0 = Length of source
;D1 = Source
;A5 = Dest

BlitterCode:
                bsr Modulu                ;set Modulu
                MOVE.L a2,$50(A0)         ;Source
                MOVE.L a1,$54(A0)         ;Dest
                MOVE.W #$09F0,$40(A0)
                MOVE.W #0,$42(A0)
                bsr StartBlit
                rts

;Blit start and wait for end of Blitter

StartBlit:
                MOVE.W d0,$dff058
BlitWait:       btst #14,$dff002
                bne.s BlitWait
                rts

;Modulu for coding set
;>= A0 = $dff000
Modulu:
                movem.l d0/a1,-(a7)
                MOVEQ #0,D0
                LEA $44(A0),A1
                MOVE.L #-1,(A1)
                LEA $62(A0),A1
                MOVE.L D0,(A1)+
                MOVE.W D0,(A1)+
                movem.l (a7)+,d0/a1
                rts


BytesBefGap:    dc.w 0
BytesAftGap:    dc.w 0
ErrorFlag:      dc.w 0
DecodeNum:      dc.w 0
DecodeAdr:      dc.l 0
FTestAdr:       dc.l 0
```

```
BlockAdr:       dc.w 0
SectNum:        dc.w 0
FirstBlock:     dc.w 0
SectBL:         dc.w 0
TrackMemory1:dc.l 0
TrackMemory2:dc.l 0
TrackMemory3:dc.l 0
TrackMemory4:dc.l 0

BlockReport1:   ds.w 11

TrackName:      dc.b 'trackdisk.device',0,0
Ende:
                END
```

Appendix C
The Deepcopy program

In a copy program, the quality of the copy is more important than the screen display. Many copy program manufacturers tend to forget this fact. This program is not very attractive, but it is fast and accurate!

Program options

F1- = Start Copy

This option starts the copying process. Set all parameters before starting. The <Esc> key can be used to terminate this option.

F2 = First Cylinder - F3 = Last cylinder

This option sets the numbers of the starting and ending cylinder to be copied. You can select this option using the cursor keys.

F4 = How many tries

This sets the number of write attempts until a verify error is output. This option is important, since not every write error is an actual defect on the disk. You can write the disk without errors if you wish. You can select this number using the cursor keys.

F5 = Write several times

This is useful if only one drive is available and several copies of one disk are needed. The program stores the data read in RAM until no more memory can be found, and then writes the data to the target disk. After the data has been written to the target disk, you can insert another target disk for writing the same information. The program doesn't need to re-read the source disk, since the information already exists in memory.

F6 = Verify Destination

This option should be used if data security is important. The data written is compared with the data read from the source disk. If an error appears during the comparison, the program attempts to rewrite the track.

F7 = Fastcopy

The option copies disks which are not copy protected. With this option several target drives can be accessed, but it cannot be used with a single drive. There are copy programs that are faster than this one, but they do not test for errors on the source disk.

F8 = Deepcopy1

This copy option permits the copying of foreign formats and some copy protected disks. The process is slow, but thorough. Deepcopy1 permits only one target drive.

F9 = Deepcopy2

The difference between Deepcopy1 and Deepcopy2 is not very great, but very useful. This uses another method for finding the track gap. In some cases copy protected disks can be copied that could not be copied under Deepcopy1.

F10 = Source drive

The source drive can be selected using this option. The number keys are used to specify the correct drive.

S = Sync correction

Normally this option should always be switched on. Only if the length of the sync mark was altered on a disk should an attempt be made to copy without sync correction. What happens during the correction is described later.

Del = Destination drive

Use this option to specify one or more target drives. The number keys are used to specify the correct drive.

Description of the copy methods

Fastcopy

The Fastcopy copy system does justice to its name. During one disk revolution a track is read, decoded and tested for errors. This is done as follows.

First a block header is loaded after which the location of the track gap and the amount of data before and after it can be determined. Then the data located before the gap is loaded, simultaneously decoded and tested for errors. This is done by reading through the DMA while the processor decodes the data. Using the blitter to decode does not make sense, since decoding cannot be faster than the loading of a track. The processor can be used more efficiently for this.

The data is compressed and stored in memory if only one drive is being used. In this case empty blocks are stored with only a few bytes.

Since the program doesn't rely on operating system functions, damaged blocks can be read and corrected before being stored. In many cases data can be saved from a damaged disk by re-copying. Only hard errors that are reported as read/write errors can be corrected in this manner.

Deepcopy

The Deepcopy option works using a completely different method. First a write attempt is made on the target disk to determine how many bytes can be written. The number of bytes which can be written depends on the number of revolutions of the drive. This can differ slightly from one drive to another. The write test is important since otherwise the copy program would not know how much data must be removed from or added to the gap. Based on this write test, first the destination and then the source disk must be inserted into the drive.

A second step is the test of the number of bytes which can be found on the source track. For this the data is read without the DMA access through the Byte Read register into memory and the number of bytes from index mark to index mark is measured. The measurement can differ by a few bytes.

A search is made for the first sync mark "by hand" and the amount of data preceding it. This provides the distance of the mark to the index mark.

Now that you know if the sync marks can be found, the track can be processed.

If at least one mark was found, the track is read through the DMA with synchronization on the first sync mark behind the index. Since the approximate number of bytes on the track has already been measured, the number can now be determined exactly.

When the length of the track is known, the number and length of the blocks, and the number of sync bytes before each block is determined. This information is used to find the track gap.

Finding the gap is done with two different criteria which can be selected with Deepcopy1 and Deepcopy2. With Deepcopy1 the gap is assumed at the end of the longest block which is valid for normal Amiga formats and also for IBM and Atari. For Deepcopy2 a search is made not for the longest block with a unique length. If a length occurs only once, it is probably the block containing a gap. The usual formats have the gap in the largest block, which is also the only one with deviating length.

Setting the sync correction is normally very important. A track of unknown construction is read in one revolution without regard to the track gaps. The gaps contain mostly undefined data which causes the controller to go out of synchronization. This prevents the following sync mark from being read properly. Since a normal sync mark consists of at least two bytes, this is not important for reading data. Problems arise only when the data must be written again. Now only the sync mark, which was not properly recognized because of the gap, is written to the target disk. An error is created on the target disk.

To combat these problems the copy program assumes that the number of sync words for every marking on a track is constant. The number of sync words which were found on the first synchronization, agrees with the actual number and provides a guide number to fit the other sync lengths, if the number of sync words following is smaller.

With this option IBM formats can also be copied. In this format a gap is found after every block which often leads to loosing the following sync words.

In every format where the number of sync words before the blocks differs, the sync correction must be switched off.

Since the number of bytes from the index to the first sync mark has been measured, it is the same on the copy. A copy protection system in which the position of the blocks against each other is tested, is copied also.

The time required for the various test procedures is well spent.

Using the program

First it must be determined if the disk is copy protected or is a foreign format (not and Amiga disk).

If during the copying of the disk with Fastcopy a read error is reported on every track, it is probably a foreign format. This disk therefore can only be copied with Deepcopy.

When a foreign format is detected, it must be decided if this is a format which was developed for the Amiga to produce faster loading. If this is the case, the disk probably can be copied with Deepcopy2 and sync correction "Off".

The reason is simple. Most fast formats for the Amiga wait for the index mark and then read the complete track. The Amiga is capable of reading a track in one revolution and the existence of more than one gap at the end of the track (directly before the index mark) would be nonsense. Therefore no sync marks can be "swallowed" because the copy program also waits for the index and therefore the gap is always at the end of the track read. The correction of the sync length is not required. The copy program with these options must also recognize the gap at the end of the track and therefore will copy it.

If a format for another computer is involved (for example IBM or Atari), try using Deepcopy1 first. The sync correction should be switched off because the position of the gap (or gaps) is not known and the sync mark may not be found.

If this is not successful, the user should try other options. Copying a copy protected disk should also be done with Deepcopy1 and sync correction switched on. If this fails, then other options can be tried.

```
;The listing of the copy program DeepCopy.s
;from the Abacus book
;Amiga Disk Drives Inside and Out
;Assembled using the AssemPro Assembler

key             = $BFEC01
Cont            = $BFEE01
IntCon          = $BFED01
MaxWait         = $4000
MinWait         = $1000
;
;------- graphics.library ------
;
Textout                = -30 - 30
InitRastPort           = -30 -168
Movee                  = -30 -210
Draw                   = -30 -216
RectFill               = -30 -276
SetAPen                = -30 -312
InitBitMap             = -30 -360
;
;--------- exec.library --------
;
ExecBase               =       4
;
AllocMem               = -30 -168
AvailMem               = -216
FreeMem                = -30 -180
OldOpenLibrary         = -30 -378
CloseLibrary           = -30 -384
FindName               = -276
;
MEMF_Chip              = $02
MEMF_Fast              = $04
MEMF_Largest           = $20000

DevList                = 350
Port                   = 36       ;Drive 0
IDNestCnt              = 294

;
;-------------------------------
;
floppy_size = floppyend - floppy

floppy_s        = floppy_size/6-1

ON              = 1
OFF             = 0

;Error-Flag Values

NoError         = $0000
NoSync          = $0001
LengthUnequal   = $0002
LengthUnequal2  = $0003
```

```
NoDisk          = $0004
ReadError       = $0005
VerifyError     = $0006
DiskProtect     = $0007
NotProtect      = $0008
Escape          = $0009

CopyAttempt1   = 3                    ;Attempts on illegal Data
CopyAttempt2   = 3                    ;Attempts on NoSync

;Load options

WithoutSync     = $0000
WithSync        = $ffff

;Write options

NoIndex         = $0000
IndexOk         = $ffff

;Size of memory used
SortBlockNum   = $40   ;Number of blocks,
                       ;whose length is sorted
Bytesread       = $3600
BSize           = 2*Bytesread
req             = 2          ;Chip-Memory
CIAA            = $BFE000

GapLengthF     = $500 ;Length of Gap for FastCopy
NumReadsF       = 5   ;Number of Read attempts for Readerror (Fast)

;Values for Cruncher

ShrtNull        = $80
MiddleNull      = $20
LongNull        = $08
ShrtNorm        = $40
MiddleNorm      = $10
LongNorm        = $04
ShrtNone        = $c0
MiddleNone      = $30
LongNone        = $02
EmptyBlock      = $01

                lea DevName,a1
                move.l $4,a6
                lea DevList(a6),a0
                jsr FindName(a6)
                move.l d0,a0
                beq Ende
                lea Port(a0),a0
                clr.w d0
\An2:           tst.l (a0)+
                beq \An1
                bset d0,Drives
\An1:           addq.w #1,d0
```

```
                cmp.w #4,d0
                bne.s \An2
        move.l ExecBase,a6
        lea    gfxname,a1
        jsr    OldOpenLibrary(a6)
        move.l d0,gfxbase
        beq    no_gfxbase
        move.l #$2800,d0
        move.l #$10002,d1
        jsr    AllocMem(a6)
        move.l d0,bit_adress
        beq    no_bitmap
        move.l #copsize+2,d0
        move.l #$10002,d1
        jsr    AllocMem(a6)
        move.l d0,cop_adress
        beq    no_copper
                MOVE.L #BSize,D0
                MOVE.L #req,D1
                JSR AllocMem(A6)
                TST.L D0
                BEQ no_DPuffer
                MOVE.L D0,TrackBuffer1
                addq.l #6,TrackBuffer1
                add.l #Bytesread-6,d0
                move.l D0,TrackBuffer2
                bsr GetMemory           ;Memory for Cruncher
                bra beg
copy_start:     MOVE.L $4,A6
                MOVE.B #$FF,$BFD300
                move.w #$0020,$dff09a
                JSR Disable

                MOVE.W #$8210,$DF096      ;set DMA-Reg.

                clr.w FreeFlagCh
                clr.w FreeFlagFa

                bsr HeadMov       ;set Heads to 0 and set Motorbits
                bsr Start_End     ;determine copy area
                cmp.b #ON,dc1     ;Deepcopy 1
                beq \ME7
                cmp.b #ON,dc2
                bne \ME5
\ME7:           move.b SD,d0              ;one Drive-Copy?
                cmp.b DD,d0
                beq \ME6                  ;yes
                bsr SwitchS
\ME5:           bsr TestProtect
                tst.l d0
                bmi \ME4
                bsr protect_source
                tst.l d0
                bmi \ME3                  ;Escape
                bra \ME5
\ME4:
```

```
                cmp.b #ON,fa            ;Fastcopy on?
                bne \ME1
                bsr FastCopy
                bra \ME3
\ME1:           cmp.b #ON,dc1           ;Deepcopy1 on?
                bne \ME2
                bsr DeepCopy
                bra \ME3
\ME2:           cmp.b #ON,dc2           ;Deepcopy2 on?
                bne \ME3
\ME6:           bsr DeepCopy
\ME3:
                bsr SwitchS
                bsr MotorOff
                bsr SwitchD
                bsr MotorOff

                move.w #$0600,$dff09e
                move.w #$8100,$dff09e   ;restore Bits
\Error:         bsr Enable
                move.w #$8020,$dff09a
Ende:           rts

TextoutL:       move.w StartTrack,d0
                lsr.w #1,d0
                move.b d0,Cylinder
                bsr reading_cyl
                rts
TextoutS:       move.w StartTrack,d0
                lsr.w #1,d0
                move.b d0,Cylinder
                bsr writing_cyl
                rts
;output Read-Error
RError:         move.w StartTrack,d0
                move.b #1,side
                btst #0,d0
                bne \RE1
                clr.b side
\RE1:           lsr.w #1,d0
                move.b d0,Cylinder
                bsr read_error
                rts
;output Write-Error
WError:         move.w StartTrack,d0
                move.b #1,side
                btst #0,d0
                bne \RE1
                clr.b side
\RE1:           lsr.w #1,d0
                move.b d0,Cylinder
                bsr write_error
                rts

FastCopy:
                bsr GapCreate
```

```
                move.b DD,d0
                cmp.b SD,d0
                beq \FC1
                bra FastCopyML          ;for several Drives
\FC1:           bra FastCopyEL          ;for one Drive

;FastCopy for several Drives
FastCopyML:
                bsr SwitchD
\FC5:           bsr TestProtect
                tst.l d0
                bpl \FC1
                bsr protect_Destination
                tst.l d0
                bmi \Error              ;Escape
                bra \FC5
\FC1:           bsr TextoutL
                bsr TrackLSF            ;load Track from Source
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                bsr SwitchD
                move.w StartTrack,d0
                bsr HeadPos
                bsr TextoutS
                bsr TrackFastWrite
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                cmp.w #DiskProtect,ErrorFlag
                bne \FC3
                bsr protect_destination
                bra \Error
\FC3:           cmp.b #ON,vd            ;Verify ON ?
                bne \FC2                ;branch if not on
                bsr TrackFVerify
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                cmp.w #VerifyError,ErrorFlag
                bne \FC2
                bsr WError
                bsr compare_drives
\FC2:           add.w #1,StartTrack
                move.w StartTrack,d0
                cmp.w EndTrack,d0
                bls \FC1
\Error          rts

;FastCopy for one Drive

FastCopyEL:
                clr.b ShrtByte          ;ShortByte for Chruncher = 0
                move.w #$1600,Length
                bsr NextMemory          ;assign memory
\FCEL1:         bsr TestProtect
                tst.l d0
                bmi \FCEL3
                bsr protect_Source
```

```
            tst.l d0
            bmi \FCEL2              ;Escape activated
            bra \FCEL1
\FCEL3:     bsr FCopy1DL            ;read in memory
            tst.l d0
            bmi \FCEL2
            move.l WriteAddrs,a5
            add.l #GapLengthF,a5
\FCEL7:     move.w TNumBufferA,StartTrack
            bsr insert_destination
            tst.l d0
            bmi \FCEL2              ;Escape
\FCEL5:     bsr TestProtect
            tst.l d0
            bpl \FCEL6
            bsr protect_Destination
            tst.l d0
            bmi \FCEL2              ;Escape activated
            bra \FCEL5
\FCEL6:     bsr FCopy1DS            ;Write Tracks
            tst.l d0
            bmi \FCEL2
            cmp.b #ON,ws            ;write repeatedly ?
            bne \FCEL8              ;no
            bsr write_b_again
            cmp.w #Escape,ErrorFlag
            beq \FCEL2
            tst.l d0
            bpl \FCEL7              ; write again
\FCEL8:     move.w StartTrack,TNumBufferA
            move.w TNumBufferE,d0
            cmp.w EndTrack,d0
            bcc \FCEL2
            bsr insert_source
            tst.l d0
            bmi \FCEL2              ;Escape
            bra \FCEL1
\FCEL2:     rts

FCopy1DL:
\FCD1:      bsr TrackLSF            ;load Track from Source
            cmp.w #NoDisk,ErrorFlag
            beq \Error
            move.l TrackBuffer2,a0  ;pass Pointer
            bsr Packe               ;crunch Track
            tst.l d0
            bmi \FCD2               ;memory full
            bsr TextoutL            ;output Text
            add.w #1,StartTrack
            move.w StartTrack,d0
            cmp.w EndTrack,d0
            bls \FCD1
\FCD2:      subq.w #1,StartTrack
            clr.l d0
            move.w StartTrack,TNumBufferE        ;Last Track
            rts
```

```
\Error:        move.l #-1,d0
               rts

;Copy portion for writing with one Drive
FCopy1DS:
               move.w StartTrack,d0      ;first Track read
\FDS1:         bsr HeadPos
               bsr TextoutS
               move.l TrackBuffer2,a0 ;Buffer for Track (Target)
               bsr EntPacke           ;Track in regular size again
               move.l a5,a1           ;Target (TrackBuffer1 + Gap)
               move.l TrackBuffer2,a0    ;Source
               move.w StartTrack,d0      ;Track to be read
               bsr CodeTrack             ;code Track
               move.w #00,FirstBlockSp   ;first Block = Null
               bsr TrackFastWrite
               cmp.w #NoDisk,ErrorFlag
               beq \Error
               cmp.w #DiskProtect,ErrorFlag
               bne \FDS3
               bsr protect_destination
               bra \Error
\FDS3:         cmp.b #ON,vd              ;Verify ON ?
               bne \FDS2                 ;branch if not on
               bsr TrackFVerify
               cmp.w #NoDisk,ErrorFlag
               beq \Error
               cmp.w #VerifyError,ErrorFlag
               bne \FDS2
               bsr WError
\FDS2:         bsr Get_Key               ;Escape activated?
               cmp.b #$45,d0
               bne \FCS4                 ;no, continue
               move.w #Escape,ErrorFlag
               bra \Error
\FCS4:         addq.w #1,StartTrack
               move.w StartTrack,d0
               cmp.w TNumBufferE,d0
               bls \FDS1
               clr.l d0
               clr.w FreeFlagCh       ;Chip-Mem is available again
               clr.w FreeFlagFa  ;Fast-Memomory is available again
               rts
\Error:        move.l #-1,d0
               rts

;crunch Track and store
;>= A0 Pointer to TrackBuffer
Packe:                                   ;Pointer to Track
               move.l a2,-(a7)
               move.l a0,a2
\PA2:          lea TrackPointer,a0       ;Pointer to Track-Table
               clr.l d0
               move.w StartTrack,d0      ;Track-Number
               lsl.w #2,d0
               adda.l d0,a0              ;Pointer to Memory
```

```
                move.l MemoryBeg,(a0)     ;store Pointer to Track
                move.l MemoryBeg,a1
                move.l a2,a0
                bsr Crunch
                tst.l d0
                bpl \PA1                  ;Ok, continue
                bsr NextMemory            ;get new Mempory
                tst.l d0
                bpl \PA2                  ;Ok, Memory obtained
\PA1:           move.l (a7)+,a2
                rts

;get Track from memory;>= A0 = Pointer to Target for Track
EntPacke:
                move.l a0,a1
                lea TrackPointer,a0       ;Pointer to Track-Table
                clr.l d0
                move.w StartTrack,d0      ;Track-Number
                lsl.w #2,d0
                adda.l d0,a0              ;Pointer to memory
                move.l (a0),a0            ;get Pointer to Track
                bsr DeCrunch
                rts

GetMemory:
                move.l a6,-(a7)
                move.l #MEMF_Chip,d1
                or.l #MEMF_Largest,d1
                move.l ExecBase,a6
                jsr AvailMem(a6)
                move.l d0,LengthChip
                bne \HS1
                clr.l MemoryChip          ;no Chip available
                bra \HS2                  ;get Fast-Memory
\HS1:           jsr AllocMem(a6)
                move.l d0,MemoryChip
\HS2:           move.l #MEMF_Fast,d1
                or.l #MEMF_Largest,d1
                jsr AvailMem(a6)
                move.l d0,LenghtFast
                bne \HS3                  ;ok
                clr.l MemoryFast
                bra \HS4                  ;no fast memory available
\HS3:           jsr AllocMem(a6)
                move.l d0,MemoryFast
\HS4:           move.l (a7)+,a6
                clr.w FreeFlagCh          ;memory is free
                clr.w FreeFlagFa
                rts

;get next memory block
NextMemory:
                tst.w FreeFlagCh          ;Chip not available
                bpl \NS1                  ;Yes, is free
\NS3:           tst.w FreeFlagFa       ;Fast memory still available
                bpl \NS4                  ;Yes, is free
```

```
\NS5:           move.l #-1,d0            ;no memory free
                bra \NS2
\NS4:           move.l MemoryFast,d0
                beq \NS5                 ;no Fast memory free
                move.l LenghtFast,d1
                move.w #$ffff,FreeFlagFa ;Fast memory occupied
                bra \NS6
\NS1:           move.l MemoryChip,d0
                beq \NS3                 ;no Chip free
                move.l LengthChip,d1
                move.w #$ffff,FreeFlagCh ;occupy Chip
\NS6:           move.l d0,MemoryBeg
                move.l d1,MemoryLength
                clr.l d0
\NS2:           rts

;load Track (Fastcopy)
;>= StartTrack = Track to be loaded
;=> SBytes = Number of Bytes to be written
;=> WriteAddrs = Address from which writing starts

TrackLSF:
                movem.l d2/a2,-(a7)
                bsr SwitchS
                move.l TrackBuffer1,a5
                move.l a5,WriteAddrs
                add.l #GapLengthF,a5
                move.l TrackBuffer2,a4
                move.w #(GapLengthF+$2ec0+2),SLength
                move.w StartTrack,d0
                bsr HeadPos
                move.w #NumReadsF-1,d2
\TSF6:          bsr FastReads
                move.w FirstBlock,FirstBlockSp  ;1. Block loaded
                cmp.w #NoDisk,ErrorFlag
                beq \TSF7
                cmp.w #ReadError,ErrorFlag
                bne \TSF5
                dbf d2,\TSF6        ;branch, if reading again
                tst.l d0            ;No Sync found
                bpl \TSF1
                move.l a4,a0             ;pass Buffer
                bsr DOSClear             ;store Track empty
\TSF1:
                bsr RError               ;output error
                move.l a4,a0
                move.l a5,a1
                move.w StartTrack,d0     ;pass Track-Number
                bsr CodeTrack       ;generate Track from data
                clr.w FirstBlockSp       ;first Block = 0
\TSF5:          clr.l d0
\TSF7:          movem.l (a7)+,d2/a2
                rts

;enter Gap Bytes in Track-Buffer
GapCreate:
```

```
                move.l TrackBuffer1,a0
                move.w #(GapLengthF/4)+4,d0
\LS1:           move.l #$aaaaaaaa,(a0)+
                dbf d0,\LS1
                rts

TrackFVerify:
                movem.l d2-d4/a4-a5,-(a7)
                clr.w d3                 ;Dest.-Counter
                clr.w VerErrFlag         ;erase Verify-Error-Flag
                move.b MotorBits,d4      ;store Motor Bits
\TF2:           move.l TrackBuffer2,a5
                move.l TrackBuffer1,a4
                add.l #GapLengthF,a4
\TF1:
                move.b tr,d2             ;number of Write attempts
                move.w d3,d1             ;Dest.-Number to D1
                bsr SwitchND             ;switch on Destination
                tst.l d0                 ;Drive present
                bmi \TF6                 ;No
\TF4:           bsr FastVerify
                cmp.w #NoDisk,ErrorFlag
                beq \TF3                 ;Error, No Disk
                cmp.w #VerifyError,ErrorFlag
                bne \TF6                 ;no Error, continue
                subq.b #1,d2             ;decrement Error-Counter
                bne \TF5                ;continue if another attempt
                bset d3,VerErrFlag       ;set Bit for Error
\TF6:           addq.w #1,d3             ;increment Dest.number
                cmpi.w #4,d3
                bcs \TF1
                bra \TF3                 ;no additional Drives
\TF5:           bsr TrackFastWrite
                bra \TF4
\TF3:           move.b d4,MotorBits
                movem.l (a7)+,d2-d4/a4-a5
                rts

TrackFastWrite:
                move.l a5,-(a7)
                move.l WriteAddrs,a5
                move.w SLength,d0
                move.w #NoIndex,d1
                bsr Writer
                move.l (a7)+,a5
                rts
DeepCopy:
                move.b DD,d0
                cmp.b SD,d0
                beq \DC1
                bra DeepCopyML           ;for several Drives
\DC1:           bra DeepCopyEL           ;for one Drive
DeepCopyEL:
                move.b #$aa,ShrtByte   ;ShortByte for Cruncher = aa
                bsr NextMemory           ;assign memory
                bsr insert_destination
```

```
                tst.l d0
                bmi \FCEL2
\FCEL9:         bsr TestProtect
                tst.l d0
                bpl \FCEL10
                bsr protect_Destination
                tst.l d0
                bmi \FCEL2              ;Escape activated
                bra \FCEL9
\FCEL10:        bsr LengthTest
                move.w StartTrack,TrackNumS
                move.w LenghtDest,d0    ;Length for Cruncher
                addi.w #$10,d0
                move.w d0,Length
\FCEL11:        bsr insert_source
                tst.l d0
                bmi \FCEL2
\FCEL1:         bsr TestProtect
                tst.l d0
                bmi \FCEL3
                bsr protect_Source
                tst.l d0
                bmi \FCEL2              ;Escape activated
                bra \FCEL1
\FCEL3:         bsr DeepCopy1DL         ;read into memory
                tst.l d0
                bmi \FCEL2
\FCEL7:         move.w TNumBufferA,StartTrack
                bsr insert_destination
                tst.l d0
                bmi \FCEL2
\FCEL5:         bsr TestProtect
                tst.l d0
                bpl \FCEL6
                bsr protect_Destination
                tst.l d0
                bmi \FCEL2              ;Escape activated
                bra \FCEL5
\FCEL6:         bsr DeepCopy1DS         ;Write Tracks
                tst.l d0
                bmi \FCEL2
                cmp.b #ON,ws            ;write several times ?
                bne \FCEL8              ;no
                bsr write_b_again
                cmp.w #Escape,ErrorFlag
                beq \FCEL2
                tst.l d0
                bpl \FCEL7              ; write again
\FCEL8:         move.w StartTrack,TNumBufferA
                move.w TNumBufferE,d0
                cmp.w EndTrack,d0
                bcs \FCEL11
\FCEL2:         rts

DeepCopy1DL:
\FCD1:          clr.w ErrorFlag
```

```
                bsr TrackLS             ;load Track from Source
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                move.l TrackBuffer2,a0  ;pass Pointer
                move.l WriteAddrs,a1
                lea -6(a0),a0
                move.l a1,(a0)
                move.w SLength,4(a0)
                bsr Packe               ;crunch Track
                tst.l d0
                bmi \FCD2               ;Memory full
                bsr TextoutL
\FCD3:          add.w #1,StartTrack
                move.w StartTrack,d0
                cmp.w EndTrack,d0
                bls \FCD1
\FCD2:          subq.w #1,StartTrack
                clr.l d0
                move.w StartTrack,TNumBufferE       ;Last Track
                rts
\Error:         move.l #-1,d0
                rts

;Copy part for writing with one Drive
DeepCopy1DS:
                move.w StartTrack,d0    ;first Track read
\FDS1:          bsr HeadPos
                bsr TextoutS
                move.l TrackBuffer2,a0  ;Buffer for Track (Target)
                lea -6(a0),a0
                bsr EntPacke            ;Track again in normal size
                move.l TrackBuffer2,a0
                move.l -6(a0),WriteAddrs
                move.w -2(a0),SLength
                move.l #$aaaaaaaa,-4(a0)
                bsr TrackWriter
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                cmp.w #DiskProtect,ErrorFlag
                bne \FDS3
                bsr protect_destination
                bra \Error
\FDS3:          cmp.b #ON,vd            ;Verify ON ?
                bne \FDS2               ;branch, when not on;
                bsr TrackFVerify
;                cmp.w #NoDisk,ErrorFlag
;                beq \Error
;                cmp.w #VerifyError,ErrorFlag
;                bne \FDS2
;                bsr WError
\FDS2:          addq.w #1,StartTrack
                move.w StartTrack,d0
                cmp.w TNumBufferE,d0
                bls \FDS1
                clr.l d0
                clr.w FreeFlagCh        ;Chip-Mem is free again
```

```
                clr.w FreeFlagFa        ;Fast-Mem is free again
                rts
\Error:         move.l #-1,d0
                rts


DeepCopyML:
                bsr SwitchD
\DC5:           bsr TestProtect
                tst.l d0
                bpl \DC3
                bsr protect_Destination
                tst.l d0
                bmi \Error              ;Escape activated
                bra \DC5
\DC3:           bsr LengthTest
                tst.l d0
                bmi \Error
\DC1:
                bsr TextoutL
                bsr TrackLS             ;load Track from Source
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                bsr SwitchD
                move.w StartTrack,d0
                bsr HeadPos
                bsr TextoutS
                bsr TrackWriter
                cmp.w #NoDisk,ErrorFlag
                beq \Error
                cmp.w #DiskProtect,ErrorFlag
                bne \DC2
                bsr protect_destination
                bra \Error
\DC2:           add.w #1,StartTrack
                move.w StartTrack,d0
                cmp.w EndTrack,d0
                bls \DC1
\Error:         rts


;load Track from Source or Destination
;>= StartTrack = Track which will be loaded

TrackLS:        bsr SwitchS
                move.l TrackBuffer1,a5
                move.w StartTrack,d0
                bsr HeadPos
                bsr TrackLoader
                rts

;Check length of Source- and Dest.-Diskette
;=>CheckLength = Length of Source-Disk
;=>LenghtDest   = Length of Dest.-Disk
LengthTest:
                bsr SwitchD
```

```
                move.w StartTrack,d0
                bsr HeadPos
                move.l TrackBuffer2,a5
                bsr erase
                move.w #Bytesread-$15,d0
                bsr Writer
                tst.l d0
                bmi \TD1                ;Disk write protected
                bsr Counter
                tst.l d0
                bmi \TD1                ;No Disk in Drive
                move.w CheckLength,LenghtDest
\TD1:           rts

;load Track after setting Motor Bits
;>= A5 = Pointer to the Read buffer
;=> WriteAddrs = Pointer to Data for writing
;=> SLength = Number of Bytes to be written

TrackLoader:
                move.w #CopyAttempt1,CopyTry1
Attempt1:
                move.w #CopyAttempt2,CopyTry2
;Attempts, on NoSync
Attempt2:

;measure length of Track (Index <=> Index)
;read data without DMA in Buffer starting at A5

                move.w #NoError,ErrorFlag
                bsr Counter        ;=> CheckLength = Length of Track
                tst.l d0            ;Disk in Drive?
                bmi \TrackLoaderEnd

;measure distance from Index to Sync
;if no Sync, then D0 = -1
;=> Syncwidth = distance to Sync

                bsr Syncdistance
                tst.l d0
                bpl \OK2
                sub.w #1,CopyTry2
                bne Attempt2

;Program part when no Sync is found

\TL7:           move.l TrackBuffer2,a5
                bsr CopyOSync
                bra \TL11

\OK2:
                move.w CheckLength,d0
                add.w #$100,d0
                move.w #WithSync,d1
                bsr loader
                move.w SyncWord,(a5)      ;store first Sync
```

```
;determine number of Bytes to Track (Sync to Sync)
;=> D0 = -1, if number deviates too much from CheckLength
;=> TrackBytes = Length of Track

               bsr SrchTEnd
               tst.l d0
               bpl \TL3
               bsr SrchTEnd2
               tst.l d0
               bpl \TL3
               sub.w #1,CopyTry1
               bne Attempt1

               move.w CheckLength,TrackBytes
               sub.w #$10,TrackBytes      ;if no End found
                                          ;shorten Track

;search for Gap if Sync was found
;>= A5 Pointer to beginning of Track
;=> Size1 = Size of the largest Block
;=> Size2 = Size of the second largest Block

;=> SizePos = Position of the largest Block
;=> SyncNum = Number of Syncs found

\TL3:
               clr.l d0
               move.w TrackBytes,d0
               add.l a5,d0
               move.l d0,EndPos

               bsr Blockidentify
               cmp.w #SortBlockNum,SyncNum
               bls \TL2                  ;Number of Sync Ok
;Too many Blocks for intermediate memory

               move.l SizePos,BegPos     ;Gap in largest Block
               bra \TL1                  ;Too many Blocks to sort
\TL2:
               bsr TrackAmiga            ;Test if Amiga-Track
               tst.l d0
               bpl \TL4                  ;branch if Amiga-Track
               cmp.b #ON,sy              ;Sync correction
               bne \TL5                  ;no
\TL4:          bsr Synccorrector
\TL5:          bsr OrderBlocks
               bsr SearchGap
\TL1:          move.l TrackBuffer2,a4    ;Destination for copying
               bsr Entirecopy
\TL11:         clr.l d0                  ;erase Error-Flag
\TrackLoaderEnd:
               rts

;load Track which has no Sync
;>= A5 = Track-Buffer
```

```
;=> GapLength = Length of Gap
;=> Syncwidth = 0  (no Sync)
;=> TrackBytes = Number of Bytes on the Track

CopyOSync:
               movem.l d2-d3,-(a7)
               move.w CheckLength,d0
               cmp.w #Bytesread-50,d0    ;Source-Track too long
               bcc \COS7
               add.w #36,d0
               bra \COS6
\COS7:         move.w #Bytesread-50,CheckLength
               move.w #Bytesread-16,d0
\COS6:         move.w #WithoutSync,d1
               bsr loader
               tst.l d0
               bmi \COS1
               move.w CheckLength,d1
               sub.w #50,d1

               move.l a5,a0              ;Track-Buffer
               move.b (a0)+,d2           ;get first Byte
\COS2:         move.b (a0)+,d3
               cmp.b d2,d3               ;compare Bytes
               bne \COS4                 ;not equal
               sub.w #1,d1               ;increase number
               bne \COS2
               bra \COSOK               ;Track the same everywhere

\COS4:         add.l #2,a0               ;jump over Gap
               move.b (a0)+,d2
\COS3:
               move.b (a0)+,d3
               cmp.b d2,d3               ;compare Bytes
               bne \COSNO                ;not always the same
               sub.w #1,d1               ;increment Counter
               bne \COS3
               move.l a5,a0

\COSOK:        move.w LenghtDest,d0
               add.w #$10,d0
\COS5:         move.b d2,(a0)+
               dbf d0,\COS5
\COSNO:        move.w LenghtDest,d0
               add.w #$4,d0
               move.w d0,TrackBytes
               move.w d0,SLength         ;number of write bytes
               move.l a5,WriteAddrs      ;beginning of Data
               move.w #NoSync,ErrorFlag
               clr.l d0
               movem.l (a7)+,d2-d3
               rts
\COS1:         move.l #-1,d0
               rts

TrackWriter:
```

```
                move.l WriteAddrs,a5
                move.w SLength,d0
                bsr Writer
                rts
erase:
                move.l TrackBuffer2,A0
                move.w #(Bytesread-$10)/4,d0
\ER2:           move.l #$aaaaaaaa,(a0)+
                dbf d0,\ER2
                rts

;determine copy area (Start- and End-Cylinder)
;>= StartTrack = Track where start is made (Track!!)
;>= EndTrack   = Track which is copied last (Track!!)

Start_End:      clr.l d0
                move.b fc,d0              ;first Cylinder
                lsl.b #1,d0               ;Cylinder => Track
                move.w d0,StartTrack
                move.w d0,TNumBufferA
                move.b lc,d0              ;last Cylinder
                lsl.b #1,d0               ;Cylinder => Track
                add.w #1,d0               ;last Track = bottom side
                move.w d0,EndTrack
                rts

;test if Disk is in the Drive
;=> D0 = -1, if no Disk in the Drive

DiskInFloppy:
                clr.l d0
                move.b $bfe001,d0
                btst #2,d0
                bne \DIF
                move.l #-1,d0
                move.w #NoDisk,ErrorFlag
\DIF:           rts

;load Track
;A5 = Pointer to Data-Buffer

Loader:         MOVEM.L d2-d3,-(A7)
                move.w d0,d3              ;read Byte num
                lsr.w #1,d3
                bsr DiskInFloppy
                tst.l d0
                bmi \L1
                MOVE.W #$8010,$DFF096     ;enable Disk-DMA
                move.w #$7f00,$DFF09E     ;erase Disk-Bits
                MOVE.L A5,A1              ;Pointer to Data-Buffer
                MOVE.L A1,$DFF020         ;pass Data-Buffer
                cmp.w #WithSync,d1
                bne \L3
                MOVE.W SyncWord,$DFF07E   ;pass SYNC-Word
                move.w #$8500,$DFF09E
                lea 2(a5),a1
```

```
                move.l a1,$DFF020       ;pass new Address
                bra \L4

\L3:            MOVE.W #$8100,$DFF09E   ;To MFM-Format
\L4:            MOVE.W #$4000,$DFF024   ;prepare transmission
                bsr Index               ;wait to Index
                tst.l d0
                bmi \L1
                or.w #$8000,d3
                MOVE.W d3,$DFF024
                MOVE.W d3,$DFF024       ;read Data
                clr.l d0                ;report Ok
                MOVE.L #$18000,D1       ;set time counter
                move.w #$0002,$dff09c   ;erase Blockdone-Int.
\L2:            MOVE.W $DFF01E,D2
                BTST #1,D2
                BNE.S \L1             ;wait for Disk-Block-Ready
                SUBQ.L #1,D1          ;decrement Counter
                BNE.S \L2             ;branch when not done
                move.l #-1,d0
\L1:            MOVE.W #$4000,$DFF024
\L5:            MOVEM.L (A7)+,D2-d3
                RTS

;wait for Index hole

Index:
                move.l d1,-(a7)
                clr.l d0
                move.l #$30000,d1
                MOVE.B $BFDD00,D0
Index1:         MOVE.B $BFDD00,D0        ;wait for Index hole
                BTST #4,D0
                Bne.s Index2
                sub.l #1,d1
                bne Index1
                move.l #-1,d0
Index2:
                move.l (a7)+,d1
                rts


;determine Motorbits, set Heads to zero
HeadMov:
                movem.l d2-d4,-(a7)
                clr.w d3
                move.b DD,d2            ;Dest.-Drives
                or.b SD,d2              ;Source-Drives
\KA1:           clr.l d0
                btst d3,d2              ;Drive now in use ?
                beq \KA2                ;not in use
                bset d3,d0              ;Bit for Drive
                lsl.b #3,d0
                eor.b #$fb,d0
                move.b d0,MotorBits
                bsr MotorOn
```

```
                clr.l d0
                bsr HeadPos
\KA2:
                addq.w #1,d3
                cmp.w #4,d3
                bne \KA1

                bsr MotBits
                move.b MotorBitsD,MotorBits
                move.b #-1,Flag
                clr.w TrackNumS
                clr.w TrackNumD
                bsr SwitchS
                movem.l (a7)+,d2-d4
                rts
TestDrive:
                move.l a0,-(a7)
                lea $bfd000,a0
                bclr #1,MotorBits
                bsr Stepper
                bset #1,MotorBits
                bsr Stepper
                move.l (a7)+,a0
                rts

;=> D0 = -1 => Disk protect
TestProtect:    bsr TestDrive
                clr.w ErrorFlag
                move.b $bfe001,d0
                btst #3,d0
                bne \TP2
                move.w #DiskProtect,ErrorFlag
                move.l #-1,d0
                bra \TP1
\TP2:           clr.l d0
\TP1:           rts

;place Head in position indicated by D0

HeadPos:        MOVEM.L D0-D3,-(A7)
                lea $bfd000,a0
                tst.w d0
                beq HeadNull
                move.w TrackNum,d2       ;current Track-Number
                CMP.W d2,D0
                BEQ.S Headend            ;End when right Track
                move.w d0,d3             ;Track-Number to D3
                move.b MotorBits,d1
                bset #2,d1               ;lower Head
                btst #0,d3
                beq Upper                ;select lower Head
                bclr #2,d1
Upper:          move.b d1,MotorBits      ;upper Head
                move.b d1,$100(a0)
                move.w d3,TrackNum
                lsr.w #1,d2
```

```
               lsr.w #1,d3            ;calculate steps
               sub.w d3,d2
               bcs.s In
               bhi.s Out
               bra Headend
In:            bclr #1,d1
               move.b d1,MotorBits
               neg.w d2
               bra.s rechok
Out:           bset #1,d1
               move.b d1,MotorBits
               bra.s rechok
Heads:         bsr Stepper
rechok:        dbf d2,Heads
Headend:       movem.l (a7)+,d0-d3
               rts

HeadNull:
               move.b Motorbits,d1
               bset #2,d1
               bset #1,d1
               move.b d1,Motorbits
               clr.w TrackNum
HeadNull1:     move.b $bfe001,d0
               btst #4,d0
               beq.s \He1
               bsr Stepper
               bra HeadNull1
\He1:          bclr #1,MotorBits
               bsr Stepper
               bset #1,MotorBits
               bsr Stepper
               bra Headend

Stepper:
               move.b MotorBits,d0
               lea $100(a0),a1
               move.b d0,d1
               bclr #0,d0
               move.b d0,(a1)
               nop
               nop
               move.b d1,(a1)
               jsr Timer
               move.b MotorBits,(a1)
               rts

;Wait loop

Timer:         MOVE.L D7,-(A7)
               MOVE.W #2500,D7
\L1:           DBRA D7,\L1
               MOVE.L (A7)+,D7
               RTS
Timer2:        MOVE.L D7,-(A7)
               MOVE.L #$6000,D7
```

```
\L1:            sub.l #1,D7
                bne \L1
                MOVE.L (A7)+,D7
                RTS

;Motor routine:   D0=0 => Motor off

MotorOff:       clr.l d0
                bra Motor
MotorOn:        move.b #$01,d0

Motor:          movem.l d1/d2,-(a7)
                lea $bfd000,a0
                tst d0
                seq d1
                andi.b #$80,d1
                move.b MotorBits,d2
                andi.b #$80,d2
                cmp.b d1,d2
                beq.s Mook
                bclr #7,MotorBits
                or.b d1,MotorBits
                move.b #$ff,d1
                eor.b d2,d1
                move.b d1,$100(a0)
                move.b MotorBits,$100(a0)
                btst #7,MotorBits
                bne Mook
                jsr Timer2
Mook:           movem.l (a7)+,d1/d2
                rts

Disable:        move.w #$4000,$dff09a
                move.l a6,-(a7)
                move.l $4,a6
                add.b #1,IDNestCnt(a6)
                move.l (a7)+,a6
                rts
Enable:         move.l a6,-(a7)
                move.l $4,a6
                sub.b #1,IDNestCnt(a6)
                bge L005
                move.w #$c000,$dff09a
L005:           move.l (a7)+,a6
                rts

;Waits for Byte during read and stores Byte (a0)+

                rts

;>= a0 = Address of data to be found
;>= A1 = Address where search starts
;>= Searchln = Number of words for error
;>= NumWords = Number of Words which is compared
;=> D1 = Number of Bytes to fund;=> Position = where found
;=> D0 = -1 = not found
```

```
;=> A0 = Position where found

Bitsrch:
            movem.l d2-d6/a2-a4,-(a7)
            move.l a0,a2
            move.l a1,a3
            clr.w d5
            move.w Searchln,d4          ;Search num
srch2:      move.w #00,d3               ;number of shift Bits
srch1:      move.l (a2),d1
            move.l (a3),d2
            bsr comp
            tst.w d0
            beq srchok
            move.l a0,a2
            tst.w d5
            beq srch4
            clr.w d5
            move.w d6,d3
            move.l a4,a3

srch4:      add.w #1,d3
            cmp.w #$10,d3
            bne srch1
srch3:      add.l #2,a3
            clr.w d5                    ;compare the right ones
            dbf d4,srch2                ;Counter for attempts
            move.l #-1,d0               ;not found
            bra srchend
srchok:
            tst.w d5
            bne srchok2
            move.l a3,a4
            move.w d3,d6

srchok2:    add.w #1,d5
            cmp.w NumWords,d5
            beq srchend1
            add.l #2,a2
            add.l #2,a3
            bra srch1
srchend1:   move.l #0,d0
            sub.w #1,d5
            lsl.w #1,d5
            suba.l d5,a3
            move.l a3,a0
            move.l a3,Position
            move.l a3,a0            ;Position
            move.w d3,BitShifts     ;Bit shifting
            sub.l a1,a3
            move.l a3,d1            ;Number of Bytes until found
srchend:    movem.l (a7)+,d2-d6/a2-a4
            rts
comp:       movem.l d1-d2,-(a7)
            lsl.l d3,d2
            swap d1
```

```
                swap d2
                eor.w d1,d2
                move.w d2,d0
                movem.l (a7)+,d1-d2
                rts


MotBits:
                move.b SD,d0            ;Source-Disk
                lsl.b #3,d0
                eor.b #$fb,d0
                move.b d0,MotorBitsS
                move.b DD,d0            ;Dest.-Disk
                lsl.b #3,d0
                eor.b #$fb,d0
                move.b d0,MotorBitsD
                rts

;switch on next Dest.-Drive
;!!! Caution !!! old Motorbits are reset
;>= D1 = which Drive ( < 4)
;=> D0 = -1, if no additional Drive available
;=> D0 = Null if OK

SwitchND:
                move.l d2,-(a7)
                clr.l d0
                move.b DD,d2
                btst d1,d2              ;Drive connected ?
                bne \SND1               ;yes, connected
                move.l #-1,d0
                bra \SND2
\SND1:          bset d1,d0              ;Bit for Drive
                lsl.b #3,d0
                eor.b #$7b,d0
                move.w StartTrack,d1
                bset #2,d0
                btst #0,d1
                beq \SND3               ;lower Head select
                bclr #2,d0
\SND3:          move.b d0,MotorBits     ;upper Head
                move.b d0,$bfd100
                clr.l d0
\SND2:          move.l (a7)+,d2
                rts

SwitchS:
                tst.b Flag
                bpl \S1
                move.b MotorBits,MotorbitsD
                move.w TrackNum,TrackNumD
                move.b MotorBitsS,MotorBits
                move.w TrackNumS,TrackNum
                move.b #$7f,$bfd100
                move.b MotorBits,d0
```

```
                bclr #7,d0
                move.b d0,$bfd100
                move.b d0,MotorBits
                clr.b Flag
\S1:            rts
SwitchD:
                tst.b Flag
                bmi \S1
                move.b MotorBits,MotorbitsS
                move.w TrackNum,TrackNumS
                move.b MotorBitsD,MotorBits
                move.w TrackNumD,TrackNum
                move.b #$7f,$bfd100
                move.b MotorBits,d0
                bclr #7,d0
                move.b d0,$bfd100
                move.b d0,MotorBits
                move.b #-1,Flag
\S1:            rts


;store Track
;>= D0 = number of Bytes to write
;>= A5 = Track-Buffer
;>= D1 = Indication if wait for Index (IndexOk/NoIndex)

Writer:      MOVEM.L D2-D3,-(A7)
                move.w d1,d2
                move.w d0,d3              ;Number to D3
                clr.w ErrorFlag
                bsr DiskInFloppy
                tst.l d0
                bmi \Protect
                move.b $bfe001,d0
                btst #3,d0
                bne \SR5
                move.w #DiskProtect,ErrorFlag
                bra \Protect
\SR5:           lsr.w #1,d3               ;from Byte make Word
                jsr MotorOn
                MOVE.W #2,$DFF09C         ;erase Disk-Block-Int.
                MOVE.L A5,$DFF020         ;pass Data-Buffer
                MOVE.W #$8210,$DFF096     ;enable Disk-DMA
                move.w #$7f00,$dff09e
                MOVE.W #$8100,$DFF09E     ;MFM-Format
                cmp.w #80,StartTrack
                bcs \SR1
                move.w #$a000,$dff09e     ;Pre-compensation
\SR1:           MOVE.W #$4000,$DFF024     ;prepare transmission
                cmp.w #NoIndex,d2
                beq \SR2
                bsr Index
                tst.l d0
                bmi \Protect
\SR2:           or.w #$c000,d3            ;switch to write
                MOVE.W d3,$DFF024
```

```
                MOVE.W d3,$DFF024         ;write Data
                clr.l d0                  ;pass OK message
                MOVE.L #$20000,D1         ;set time counter
\SR3:           MOVE.W $DFF01E,D2
                BTST #1,D2
                BNE.S \SR4            ;wait for Disk-Block-Ready
                SUBQ.L #1,D1          ;decrement Counter
                BNE.S \SR3            ;branch when not done
\Protect:       move.l #-1,d0
\SR4:           move.w #$4000,$dff024
                MOVEM.L (A7)+,D2-D3
                RTS
;Searches for Track-End when Sync found
;>= A5 = Pointer to Track-Buffer (Sync found)
;=> TrackBytes = Number of Bytes on the Track (seek for Sync)
;=> D0 = -1, too much deviation from CheckLength

SrchTEnd:
                move.l a5,a0
                clr.l d0
                move.w CheckLength,d0
                sub.w #$4,d0
                adda.l d0,a0
                move.w #$10,d0
                bsr SrchSync
                tst.l d0
                bmi \STE2
                suba.l a5,a0
                sub.w #2,a0
                move.w a0,TrackBytes
                rts
\STE2:          move.l #-1,d0
                move.w #LengthUnequal,ErrorFlag
                rts

SrchTEnd2:
                move.l a5,a0
                adda.l #$04,a0
                move.l a0,a1
                clr.l d0
                move.w CheckLength,d0
                sub.w #$40,d0
                adda.l d0,a1
                move.w #$40,Searchln
                move.w #$60,NumWords
                bsr Bitsrch
                tst.l d0
                bmi \STE2
                suba.l a5,a0
                suba.l #6,a0              ;Position before Sync
                move.w a0,TrackBytes
                rts
\STE2:          move.l #-1,d0
                move.w #LengthUnequal2,ErrorFlag
                rts
```

```
;search for Blocks and store
;>= A5 = Pointer to Track beginning
;=> Size1 = Size of the largest Block
;=> Size2 = Size of the second largest Block
;=> SizePos = Position of the largest Block
;=> SyncNum = Number of Syncs found
;=> Blocks = Buffer in which the Block sizes are stored
Blockidentify: movem.l d2-d4/a3,-(a7)
               move.w TrackBytes,d2      ;Bytes on Track
               move.l a5,a3              ;Beginning of Track
               clr.w Size1
               clr.w Size2
               clr.w SyncNum
               clr.w d3

\S2:           move.w d2,d0
               move.l a3,a0              ;where to start search
               bsr SrchSyncF
               tst.l d0
               bmi \S5                   ;End
               tst.w d1
               beq \S6
               move.w d3,d4
               add.w d1,d3               ;Number + Sync
               bsr Blockentry
               bsr Size
               clr.w d3                  ;Sync gap = 0
\S6:
               add.w #2,d3               ;Sync gap +2
               add.w #2,d1
               andi.l #$ffff,d1
               adda.l d1,a3
               sub.w d1,d2
               bcc \S2
               bra \S7
\S5:           move.w d2,d1
               add.w #2,d1               ;Number +2
               move.w d3,d4              ;Sync message
               add.w d1,d3               ;Number + Syn
               bsr Blockentry
               bsr Size
\S7:           movem.l (a7)+,D2-d4/A3
               rts

;Enter size of Blocks
;>= D1 = Block size

Size:
               cmp.w Size1,d3
               bcs \S3
               move.w Size1,Size2
               move.w d3,Size1
               move.l a0,a1
               move.l a1,SizePos
               bra \S4
\S3:           cmp.w Size2,d3
```

```
                bcs \S4
                move.w d1,Size2
\S4:            rts

Blockentry:
                move.w SyncNum,d0
                cmp.w #$40,d0          ;Too many Blocks ?
                bcc \S1                ;yes, do not store
                lsl.w #2,d0
                lea Blocks,a1
                lsr.w #1,d4
                move.w d4,(a1,d0)
                add.w #2,d0
                move.w d3,(a1,d0)      ;store Block
\S1:            add.w #1,SyncNum
                rts

;seek Sync-Mark (fast)
;>= A0 = Search address
;>= D0 = Byte number for errors permitted
;=> A0 = Sync address
;=> D0 = -1, no Sync found
;=> D1 = found after xx Bytes
SrchSyncF:
                lsr.w #1,d0            ;from Byte make Word
                move.l a0,a1
\SSF2:          cmp.w #$4489,(a0)+
                beq \SSF1
                dbf d0,\SSF2
                move.l #-1,d0
                bra \SSF3
\SSF1:          suba.l #2,a0
                move.l a0,d0
                sub.l a1,a0
                move.w a0,d1
                move.l d0,a0
                clr.l d0
\SSF3:          rts
;search for Sync-Mark
;>= A0 = Search address
;>= D0 = Byte number for errors permitted
;=> A0 = Sync address
;=> D0 = -1, no Sync found
;=> D1 = found after  xx Bytes
;=> BitShifts = shifted by xx Bits



SrchSync:
                movem.l d2-d4/a2,-(a7)
                move.l a0,a2
                lsr.w #2,d0        ;Byte out, with longword value
                lea SyncBase,a1
\SS3:           clr.l d1
                move.l #$ffff0000,d3
                move.l (a0)+,d2
```

```
\SS2:           move.l d2,d4
                and.l d3,d4
                cmp.l (a1,d1),d4
                beq \SS1
                add.w #4,d1
                lsr.l #1,d3
                cmp.w #$40,d1
                bls \SS2
                dbf d0,\SS3
                move.l #-1,d0
                bra \SS4
\SS1:           clr.l d0
                lsr.w #2,d1
                move.w d1,BitShifts
                cmp.w #$8,d1
                bcc \SS5
                suba.l #2,a0
\SS5:           suba.l #2,a0
                move.l a0,d1
                suba.l a2,a0
                exg.l a0,d1
\SS4:           movem.l (a7)+,d2-d4/a2
                rts

;Distance Index <=> Sync
;=> Syncwidth = Distance Index <=> Sync
;=> D0 = -1, no Sync found

Syncdistance:
                move.l a5,a0
                move.w CheckLength,d0
                bsr SrchSync
                tst.l d0
                bpl \SE1
                move.w #NoSync,ErrorFlag
                bra \SE2
\SE1:
                move.w d1,Syncwidth
\SE2:           rts

;A5 = TrackBuffer

;Counter of Data on Disk without Sync and without DMA
;>= A5 = Pointer to Verify-Buffer for reading Data without Sync
;=> CheckLength = Length of a Track (From Index to Index)

Counter:
                bsr DiskInFloppy
                tst.l d0
                bmi \Z5
                movem.l d2-d4/a2-a3,-(a7)
                move.w #$0600,$dff09e     ;switch Sync off
                move.w #$8000,$dff024
                lea $dff01b,a0
                lea $dff01a,a1
                lea $bfdd00,a2
```

```
                move.l #15,d2
                move.l #4,d3
                move.w #1,d4
                clr.w d0
                move.b (a2),d1          ;erase Byte-Ready-Flag
\Z4:            move.b (a2),d1
                btst #4,d1
                beq.s \Z4
                move.w (a1),d1          ;erase Byte-Ready-Flag
                move.b (a2),d1          ;erase Indexbit
                move.l a5,a3
\Z1:
                btst d2,(a1)
                beq.s \Z1
                btst d3,(a2)
                bne.s \Z2
                add.w d4,d0
                move.b (a0),(a3)+
                bra \Z1
\Z2:
                andi.w #$fffe,d0
                move.w d0,CheckLength
                cmp.w #Bytesread,d0     ;Track too long
                bcs \Z6
                move.w #Bytesread-16,CheckLength
\Z6:            bclr #31,d0             ;erase Errorbit
                movem.l (a7)+,d2-d4/a2-a3
\Z5:            move.w #$4000,$dff024
                rts

;Sort Block sizes

OrderBlocks:
                move.l d2,-(a7)
                cmp.w #SortBlockNum,SyncNum
                bhi \OBEND              ;too many Blocks
                tst.w SyncNum
                beq \OBEND              ;no Blocks
                clr.w NumSortBlock
                lea Blocks,a0
                move.w SyncNum,d0
\OB5:           move.l (a0)+,d1
                sub.w #1,d0
                lea SortBlocks,a1       ;Number of Blocks -1
                move.w NumSortBlock,d2
                tst.w d2
                beq \OB2                ;first Block
\OB4:           cmp.w (a1)+,d1
                bne \OB3                ;found
                add.w #1,(a1)
                bra \OB6
\OB3:           adda.w #2,a1
                sub.w #1,d2
                bne \OB4                ;continue search
\OB2:           move.w d1,(a1)+
                move.w #1,(a1)+
```

```
                add.w #1,NumSortBlock
\OB6:           tst.w d0
                bne \OB5                    ;new search
\OBEND:         move.l (a7)+,d2
                rts

SearchGap:
                move.l d2,-(a7)
                clr.l BegPos
                move.w NumSortBlock,d0

                cmp.b #1,dc1
                beq \SL10       ;DeepCopy 1 (Gap after large Block)

;search for single Blocks

                sub.w #1,d0                 ;Block number -1
\SL3:
                bsr SingleBlock             ;search for single Block
                tst.l d0                    ;Gap found ?
                bpl \SL7                    ;branch when not found
\SL10:          move.l SizePos,a0           ;Gap with largest Block
\SL7:
                move.l A0,BegPos
                cmp.l EndPos,a0
                bcs \SL8
                move.l a5,a0
                move.l a0,BegPos
\SL8:
                move.l SizePos,d0
                cmp.l EndPos,d0
                bcs \SL9
                move.l a5,SizePos
\SL9:           move.l (a7)+,d2
                rts


;Search for single Block
;=> D0 = Block number
;=> D0 = Block number of next Block
;=> D0 = -1, when none found
;=> D1 = Size of Block
;=> a0 = Address of the Blocks

SingleBlock:
                movem.l d2-d4,-(a7)
                and.l #$ffff,d0             ;erase error message
                tst.w d0
                bmi \EB3
                lea SortBlocks,a0
\EB2:           move.w d0,d1
                lsl.w #2,d1
                move.l (a0,d1),d2
                cmp.w #1,d2
                beq \EB1                    ;Block found
```

```
                sub.w #1,d0
                tst.w d0
                bpl \EB2
\EB3:           move.l #-1,d0
                bra \EBEND                 ;Error
\EB1:           move.w (a0,d1),d2          ;Block length
                clr.w d3                   ;erase Blockadr.
                lea Blocks,a0
\EB4:           move.l (a0)+,d1
                add.w d1,d3                ;determine Address
                cmp.w d1,d2
                bne \EB4                   ;continue if not done
                sub.l a0,a0                ;clear A0
\EB5:           move.w d3,a0               ;Offset
                add.l a5,a0                ;Address
                move.w d2,d1               ;Block size
                sub.w #1,d0                ;Number of next Block
\EBEND:         movem.l (a7)+,d2-d4
                rts

;Test if Track on Amiga-Format
;=> D0 = Null, if Track Amiga-Format
;=> D0 = -1, if not Amiga-Format
TrackAmiga:
                move.l d2,-(a7)
                lea Blocks,a0           ;Pointer to Block storage
                cmp.w #$0b,SyncNum      ;SyncNum for Amiga-Format
                bne \PL1                ;No not Amiga-Format
                clr.w d1
                clr.l d2
                move.w #$0a,d0
\PL3:           move.w d0,d2
                lsl.w #2,d2
                cmp.w #$0440,2(a0,d2)     ;Block length for Amiga
                bne \PL2
                add.w #1,d1
\PL2:           dbf d0,\PL3
                cmp.w #$9,d1              ;at least 9 Amiga-Blocks
                bcs \PL1
                clr.l d0                  ;OK, Amiga-Format
                move.b #ON,AmigaTrack
\PL4:           move.l (a7)+,d2
                rts
\PL1:           move.l #-1,d0             ;not Amiga-Format
                move.b #OFF,AmigaTrack
                bra \PL4
;The program assumes that all Syncs are equal in length
;all Syncs are fitted to the first
;>= A5 = Pointer to Track-Buffer

Synccorrector:
                movem.l d2-d4,-(a7)
                cmp.w #1,SyncNum
                beq \SK1                  ;only one Sync
                lea Blocks,a0
                clr.w d0
```

```
                clr.l d3
                move.l a5,a1
                move.l (a0)+,d2
                move.w d2,d3              ;Block length
                swap d2                   ;Sync value
\SK3:           adda.l d3,a1              ;Pointer to Block
                add.w #1,d0
                cmp.w SyncNum,d0
                bcc \SK1
                move.l (a0)+,d1
                move.w d1,d3              ;Block length
                swap d1                   ;SyncNum from Block
                cmp.w d2,d1
                bcc \SK3                  ;do not correct
                sub.w d2,d1
                not.w d1
                move.w d1,d4              ;Number of new Syncs
                move.l a1,-(a7)
\SK2:
                cmp.l SizePos,a1          ;must Pos be changed
                bne \SK8
                sub.l #2,SizePos          ;change SizePos
\SK8:           move.w #$4489,-(a1)

                dbf d4,\SK2
                move.l (a7)+,a1
                add.w #1,d1
                add.w d1,-4(a0)
                lsl.w #1,d1
                sub.w d1,-6(a0)
                add.w d1,-2(a0)
                bra \SK3
\SK1:           cmp.w #LengthUnequal,ErrorFlag
                beq \SK7                 ;no end correction required
                move.w d2,d0             ;Sync value
                move.l EndPos,a1
                add.l #2,a1              ;set Pointer to Sync
                bra \SK4
\SK6:           cmp.w #$4489,(a1)+    ;test transition to beginning
                bne \SK5
\SK4:           dbf d0,\SK6
\SK7:           movem.l (a7)+,d2-d4
                rts
\SK5:           sub.w d0,d2
                lsl.w #1,d2
                sub.l #2,EndPos
                sub.w #2,TrackBytes
                sub.w #2,-2(a0)           ;Last Block is shorter
                bra \SK7

;after the Gap has been determined, the Data are copied;and
written
;>= A5 = TrackBuffer (Source)
;>= A4 = TrackBuffer (Destination)

Entirecopy:
```

```
                move.l a2,-(a7)
                move.l a4,a1
                move.l BegPos,a0
                sub.l a5,a0                ;Offset of the beginning
                clr.l d0
                move.w Syncwidth,d0
                sub.w #$0044,d0            ;distance correction
                bcc \UK6                   ;distance too small ?
                clr.w d0                   ;yes, distance = 0
\UK6:
                add.l d0,a0                ;distance from Index
                move.w a0,Offset          ;number of Bytes before Sync
                suba.l a0,a1               ;beginning of writing
                move.l a1,WriteAddrs
                move.w TrackBytes,d0
                lsr.w #1,d0                ;Byte to Word
                move.w LenghtDest,d1
                move.l BegPos,a0           ;Source-Address
                move.l a4,a1               ;Destination-Address
                cmp.b #ON,AmigaTrack
                bne \UK9
                add.l #4,a1
\UK9:           move.l #$aaaaaaaa,-4(a1)
\UK4:           move.w (a0),(a1)+
                sub.w #1,d1
                cmp.l EndPos,a0
                bcs \UK3
                move.l a5,a0
                bra \UK5
\UK3:           adda.l #2,a0
\UK5:           dbf d0,\UK4
                move.l (a7)+,a2
                and.w #$f000,d1
                bne \UK8              ;Source longer than Destination
\UK7:           move.w #$aaaa,(a1)+ ;if Dest. is longer than Source
                dbf d1,\UK7
\UK8:           move.w LenghtDest,d0
                sub.w #$0006,d0
                add.w Offset,d0
                move.w d0,SLength
                rts
;read Track and decode
;>= A4 = Pointer to Buffer for decoded Data
;>= A5 = Pointer to Buffer for coded Data

FastReads:
                MOVEM.L D2-D4/a3/a6,-(A7)
                lea decode,a3               ;Jump point for decode
                move.w #$080,DecodeCnt      ;number of
                                            ;Longwords to decode
                lea $40(a5),a0
                move.l a0,DecodeAdr         ;Data area of first Block
                adda.l #$400,a0
                move.l a0,FTestAdr        ;Address of the next Block
                move.w #OFF,VerifyFlag
                bra Fread
```

```
FastVerify:     movem.l d2-d4/a3/a6,-(a7)
                lea FVerify,a3              ;Jump point for Verify
                move.l a5,DecodeAdr
                lea $440(a5),a0
                move.l a0,FTestAdr
                move.w #ON,VerifyFlag

Fread:
                clr.w ErrorFlag
                bsr DiskInFloppy
                tst.l d0
                bmi \FastError
                MOVEA.L A5,A6               ;Track-Buffer
                move.l #$aaaaaaaa,(a6)+
                move.w #$4489,(a6)+         ;enter first Sync
                bsr search
                tst.l d0
                bmi \FreadEnd

                bsr FErase                  ;prepare Track-Buffer
                clr.l d2
                move.w BytesBefGap,d2       ;display Byte before Gap
                tst.l d2
                beq \FL1                    ;No Byte before Gap
                clr.w BlockAdr              ;Offset in Block
                bsr Readbyte                ;read Byte
                clr.l d0
                move.w BytesBefGap,d0
                move.l a5,a6
                adda.l d0,a6                ;Pointer to next Buffer
                move.l #$aaaaaaaa,(a6)+
                move.w #$4489,(a6)+         ;enter first Sync
\FL1:           move.w BytesAftGAp,d2
                tst.l d2
                beq \FL2
                clr.w BlockAdr
                bsr Readbyte
\FL2:           cmp.w #ON,VerifyFlag
                beq \FL3
                bsr lastblock       ;no last Block during Verify
\FL3:           BTST #2,$BFE001
                BEQ.S \FastError            ;Error, if no Disk
                MOVEQ #0,d0                 ;OK-Message
                move.l #$aaaaaaaa,$2ec0(a5)
                                            ;create Gap after Data
                bra \FreadEnd
\FastError:    move.l #-1,d0
\FreadEnd:     MOVEM.L (A7)+,D2-D4/a3/A6
                RTS

;prepare Track-Buffer (erase Block starts)
;>= A5 = Pointer to Track-Buffer

FErase:         move.l a5,a0
                move.w #10,d1
```

```
                clr.l d0
\L1:            move.l d0,$440(a0)
                adda.l #$440,a0
                dbf d1,\L1
                lea BlockMessage,a0
                move.w #10,d1
\L2:            clr.w (a0)+
                dbf d1,\L2
                rts

;read number of Bytes selected
;>= A6 = Pointer to Destination
;>= D2 = Number of Bytes to be read

Readbyte:
                jsr install
                MOVE.W D2,D0
                LSR.W #1,D0
                ORI.W #$8000,D0
                add.w #1,d0
                MOVE.W D0,36(A1)
                MOVE.W D0,36(A1)
                jsr (a3)
                LEA $DFF000,A1
                MOVE.W #$4000,$24(A1)
                rts

;preparation for reading
;>= A6 Pointer to Track-Buffer

install:        LEA $DFF000,A1
                move.w #$4000,$24(a1)     ;reset Disk-Len
                move.w #$8400,$9e(a1)     ;switch on Disk-Sync
                move.w #$4489,$7e(a1)     ;SYNC-Mark
                MOVE.L A6,$20(A1)         ;pass Buffer
                move.w #$0002,$dff09c
                rts

;code Longword and store in Buffer
;>= D0 = Longword
;>= A0 = Pointer to Buffer

CodeLWord:      MOVEM.L D2-D3,-(A7)
                MOVE.L D0,D3
                LSR.L #1,D0
                BSR \CH1
                MOVE.L D3,D0
                BSR \CH1
                BSR SetBorders
                MOVEM.L (A7)+,D2-D3
                RTS
\CH1:           ANDI.L #$55555555,D0
                MOVE.L D0,D2
                EORI.L #$55555555,D2
                MOVE.L D2,D1
                LSL.L #1,D2
```

```
                LSR.L #1,D1
                BSET #$1F,D1
                AND.L D2,D1
                OR.L D1,D0
                BTST #0,-1(A0)
                BEQ.S \CH2
                BCLR #$1F,D0
\CH2:           MOVE.L D0,(A0)+
                RTS
;set borders properly

SetBorders:     MOVE.B (A0),D0
                BTST #0,-1(A0)
                BNE.S \CH4
                BTST #6,D0
                BNE.S \CH6
                BSET #7,D0
                BRA.S \CH5
\CH4:           BCLR #7,D0
\CH5:           MOVE.B D0,(A0)
\CH6:           RTS

;determine Checksum for Data area
;>= D1 = number of Bytes (must always be divisible by 4)
;>= A0 = Pointer to Buffer
;=> D0 = Checksum

TestSum:        MOVE.L D2,-(A7)
                LSR.W #2,D1
                SUBQ.W #1,D1
                MOVEQ #0,D0
\PS1:           MOVE.L (A0)+,D2
                EOR.L D2,D0
                DBRA D1,\PS1
                ANDI.L #$55555555,D0
                MOVE.L (A7)+,D2
                RTS

;decode Block-Header
;>= A0 is Pointer to Header
;=> D0 = Header

Header:         move.l (a0)+,D0
                move.l (a0)+,D1
                andi.l #$55555555,d0
                andi.l #$55555555,d1
                lsl.l #1,D0
                or.l D1,D0
                rts

;find first undisturbed Block
;=> A6 = Pointer to Track-Buffer
;=> D0 = Null: Block found
;=> BytesBefGap = Number of Bytes before the Gap
;=> BytesAftGAp = Number of Bytes after the Gap
```

```
search:
            movem.l d2-d4,-(a7)
            move.w #11,d2           ;Number of errors permitted
\SU1:       bsr install
            move.w #$8024,d0          ;read $24-Words
            MOVE.W D0,$dff024
            MOVE.W D0,$dff024
            bsr Blockready            ;wait for Block-Ready
            tst.l d0                  ;Error, then D0 = -1
            bmi \SUError

            lea 8(a5),a0              ;Pointer to Block-Header
            moveq #$28,d1             ;number of Long words
            bsr TestSum               ;Sum for Header
            move.l d0,d3              ;save Sum
            lea 48(a5),a0             ;*Sum
            bsr Header                ;get Sum from Header
            cmp.l d0,d3               ;compare Sums
            bne \SUNeu
            lea 8(a5),a0
            jsr Header                ;decode Header
            move.w d0,d3              ;Header to D3
            lsr.w #8,d3
            andi.w #$00ff,d3          ;isolate Sector number
            addi.w #1,d3              ;increment Sector number
            cmp.w #$000a,d3           ;Number bigger than 10?
            bls \SU2                  ;No, OK
            clr.w d3                  ;Number = 0
\SU2:       move.w d3,SectNum         ;store Number
            move.w d3,FirstBlock      ;Number first Block

            move.w d0,d3              ;Header
            andi.w #$ff,d3            ;Sectors to Gap
            cmp.b #$0c,d3             ;Header OK?
            bcs.s \SUok
\SUNeu:     dbf d2,\SU1
            bra \SUError
\SUok:

            sub.w #1,d3               ;number of Blocks to Gap
            move.w d3,d2
            move.w #$000b,d4
            sub.b d2,d4             ;Number of Blocks after Gap
            mulu #$440,d3             ;Number of Bytes to Gap
            mulu #$440,d4             ;Number of Bytes after
Gap
            clr.l d0
            move.w d3,BytesBefGap
            move.w d4,BytesAftGAp
            move.w #$0b,SectBL;Sectors before Gap after loading
            bra \SUEnd
\SUError:   move.l #-1,d0
            move.w #ReadError,ErrorFlag
\SUEnd:     movem.l (a7)+,d2-d4
            rts
```

```
Blockready:     clr.l d0                    ;erase Error-Flag
                move.l #$20000,d1
                move.w #$0002,$dff09c       ;erase Disk-Int.
\B1:            MOVE.W $DFF01E,D0
                BTST #1,D0
                bne.s \B2
                sub.l #1,d1
                bne \B1
                move.l #-1,d0               ;Error occurred
\B2:            RTS


;decode Bytes until Block has been read

decode:
                movem.l d2-d3/a2,-(a7)
                clr.l d3
                move.w BlockAdr,d3          ;Offset in Block
                move.l FTestAdr,a0          ;Address for testing if
                                            ;Block already loaded
                move.l DecodeAdr,a2         ;Address where
                                            ;decoding is done
                move.w DecodeCnt,d2         ;Number for decoding

\DC1:           MOVE.W $DFF01E,D0
                BTST #1,D0                  ;area already read
                bne  \DCEnd                 ;Yes, End
                tst.l (a0)                  ;TestAdr
                beq \DC1              ;wait until Block has been read
                movem.l a0-a1,-(a7)         ;save Register
                lea -$40(a2),a1             ;* Block start
                bsr BlockCheck              ;check Block
                movem.l (a7)+,a0-a1         ;restore Register
                move.w SectNum,d0
                mulu #$200,d0
                move.l a4,a1       ;Base address for destination data
                add.l d0,a1                 ;Address of the Block
\DC2:
                MOVE.W $DFF01E,D0
                BTST #1,D0
                bne.s \DCEnd                ;area already read

                move.l (a2),D0
                move.l $200(a2),D1
                adda.l #4,a2
                andi.l #$55555555,d0
                andi.l #$55555555,d1
                lsl.l #1,D0
                or.l D1,D0
                move.l d0,(a1,d3)           ;enter Longword
                addq.w #4,d3
                subq.w #1,D2                ;Decode number
                bne \DC2
                adda.l #$240,a2             ;increment Address
                adda.l #$440,a0             ;TestAdr
                move.l #$080,D2             ;Decode number
```

```
                clr.w d3                ;Offset to Null
                add.w #1,SectNum        ;increment Sector number
                cmp.w #$0b,SectNum      ;Number more than 10?
                bcs \DC3                ;No, OK
                clr.w SectNum           ;Number = 0
\DC3:           bra \DC1

\DCEnd:
                move.w d3,BlockAdr
                move.l a2,DecodeAdr
                move.l a0,FTestAdr
                move.w D2,DecodeCnt

                movem.l (a7)+,d2-d3/a2
                RTS

;decode last Block

Lastblock:
                movem.l d2-d3/a2,-(a7)

                move.w SectNum,d0
                mulu #$200,d0
                move.l a4,a1            ;Base address for dest. data
                add.l d0,a1             ;Address of the Block
                clr.l d3
                move.l DecodeAdr,a2
                move.w DecodeCnt,d2
\LB1:           move.l (a2),D0
                move.l $200(a2),D1
                adda.l #4,a2
                andi.l #$55555555,d0
                andi.l #$55555555,d1
                lsl.l #1,D0
                or.l D1,D0
                move.l d0,(a1,d3)
                addq.w #$4,d3
                subq.w #1,D2            ;Decode number
                bne \LB1
                movem.l (a7)+,d2-d3/a2
                               RTS

;test Block for Error
;A1 = Pointer to start of Block

BlockCheck:
                movem.l d2-d3,-(a7)
                clr.l d3
                move.w SectNum,d3
                lsl.w #1,d3             ;Sector number => Offset
                lea BlockMessage,a0
                move.w (a0,d3),d0       ;get entry
                tst.w d0                ;already tested ?
                bne \CBEnd2             ;yes, End

                lea 64(a1),a0
```

```
                move.w #$400,d1
                jsr TestSum              ;Sum for Data block
                move.l d0,d2             ;save Sum
                lea 56(a1),a0            ;Pointer to Data sum
                jsr Header               ;decode Sum
                cmp.l d0,d2
                bne \DataFalse

                lea 8(a1),a0
                bsr Header               ;decode Header
                move.w d0,d2             ;store lower Word
                lsr.w #8,d2              ;Sector number to d2
                cmp.b SectNum+1,d2       ;right Sector
                bne \FalseSector
                swap d0                  ;Track-Number to D0
                cmp.b StartTrack+1,d0    ;right Track?
                bne \FalseTrack
                andi.l #$ff00,d0
                cmp.w #$ff00,d0
                bne \NotDOSTrack
                lea 8(a1),a0
                moveq #$28,d1            ;number of Longwords
                bsr TestSum              ;Sum for Header
                move.l d0,d2             ;save Sum
                lea 48(a1),a0            ;*Sum
                bsr Header               ;get Sum from Header
                cmp.l d0,d2              ;compare Sums
                bne \HeaderFalse
                move.w #$ffff,d0
\CBEnd1:        lea BlockMessage,a0
                move.w d0,(a0,d3)
                btst #0,-1(a1)
                beq \CB1
                move.l #$2aaaaaaa,(a1)
                bra \CB2
\CB1:           move.l #$aaaaaaaa,(a1)
\CB2:           move.l #$44894489,4(a1)
                move.w #$ff00,d0         ;create new Header
                move.b StartTrack+1,d0
                swap d0
                move.b SectNum+1,d0
                lsl.w #8,d0
                move.b SectBL+1,d0
                lea 8(a1),a0
                bsr CodeLWord            ;store Header
                lea 8(a1),a0
                moveq #$28,d1            ;Longword number
                bsr TestSum              ;Sum for Header
                lea 48(a1),a0            ;*Sum
                bsr CodeLWord            ;store Checksum
                subq.w #1,SectBL
\CBEnd2:        movem.l (a7)+,d2-d3
                rts

\FalseSector: move.w #$0001,d0
                  bra \Flagset
```

```
\FalseTrack:      move.w #$0002,d0
                  bra \Flagset
\NotDOSTrack:     move.w #$0017,d0
                  bra \Flagset
\HeaderFalse:     move.w #$001b,d0
                  bra \Flagset
\DataFalse:       move.w #$0019,d0
\Flagset:         move.w #ReadError,ErrorFlag
                  bra \CBEnd1

;compare Bytes read with old
;>= A4 = Pointer to Base address of the old Block

FVerify:
                cmp #VerifyError,ErrorFlag
                beq \FVEnd2
                movem.l d2-d3/a2,-(a7)
                clr.l d3
                move.w BlockAdr,d3          ;Offset in Block
                move.l FTestAdr,a0          ;Address for test if
                                            ;Block already loaded
                move.l DecodeAdr,a2         ;Address, where
                                            ;comparison is made
\FV1:           MOVE.W $DFF01E,D0
                BTST #1,D0                  ;area already read
                bne  \FVEnd                 ;Yes, End
                tst.l (a0)                  ;TestAdr
                beq \FV1                   ;Wait until Block was read
                move.w #$110,d2             ;comparison number
                move.w SectNum,d0
                sub.w FirstBlockSp,d0
                bcc \FV2
                addi.w #11,d0
\FV2:
                mulu #$440,d0
                move.l a4,a1             ;Base address for dest. data
                add.l d0,a1               ;Address of the Block
\FV3:
                move.l (a2)+,d0
                cmp.l (a1,d3),d0
                beq \FV6                    ;Verify Ok
;test for special case ($aaaaaaaa and $2aaaaaaa)
                move.l (a1,d3),d1
                eor.l d1,d0
                cmp.l #$80000000,d0
                bne \FV5
\FV6:           addq.w #4,d3
                subq.w #1,d2
                bne \FV3
                adda.l #$440,a0             ;TestAdr
                clr.l d3                    ;Offset to Null
                add.w #1,SectNum            ;increment Sector number
                cmp.w #$0b,SectNum          ;Number higher than 10?
                bcs \FV1                    ;No, OK
                clr.w SectNum               ;Number = 0
                bra \FV1
```

```
\FV5:          move.w #VerifyError,ErrorFlag
               bra \FVEnd3
\FVEnd:
               clr.w ErrorFlag
               move.w d3,BlockAdr
               move.l a2,DecodeAdr
               move.l a0,FTestAdr
\FVEnd3:       movem.l (a7)+,d2-d3/a2
\FVEnd2:       RTS

;code Track
;>= A0 = Pointer to Source
;>= A1 = Pointer to Destination
;>= D0 = Track-Number
CodeTrack:
               movem.l d2-d4/a2-a3,-(a7)
               move.l d0,d4
               move.l a0,a2
               move.l a1,a3
               moveq #$0b,d2
               clr.w d3
\CT1:          move.w #$ff00,d0
               move.b d4,d0
               swap d0
               move.w d3,d0
               lsl.w #8,d0
               move.b d2,d0
               move.l a2,a0             ;Source
               move.l a3,a1             ;Destination
               bsr ConstructBlock
               add.l #$440,a3
               add.l #$200,a2
               addq.w #1,d3
               subq.w #1,d2
               bne \CT1
               movem.l (a7)+,d2-d4/a2-a3
               rts

;code Block and create Header
;>= A0 = Uncoded Data (Source)
;>= A1 = Data buffer for coded Data (Destination)
;>= D0 = uncoded Header

ConstructBlock: MOVEM.L D2/A2/A4,-(A7)
               MOVEA.L A1,A4
               MOVEA.L A0,A2
               MOVE.L D0,D2
               MOVEQ #0,D0
               LEA 0(A4),A0
               BSR CodeLWord
               MOVE.L #$44894489,4(A4)
               MOVE.L D2,D0
               LEA 8(A4),A0
               BSR CodeLWord
               MOVEQ #3,D2
\EB1:          MOVEQ #0,D0
```

```
                BSR CodeLWord
                DBRA D2,\EB1
                LEA 8(A4),A0
                MOVEQ #$28,D1
                BSR TestSum
                LEA $30(A4),A0
                BSR CodeLWord
                MOVE.L #$200,D0
                MOVEA.L A2,A0
                LEA $40(A4),A1
                BSR CodeBlock
                LEA $40(A4),A0
                MOVE.W #$400,D1
                BSR TestSum
                LEA $38(A4),A0
                BSR CodeLWord
                MOVEM.L (A7)+,D2/A2/A4
                RTS

;code Data Block
;>= D0 = Length of the Source
;>= A0 = Pointer to Source
;>= A1 = Pointer to Destination

CodeBlock:

                movem.l  d2/a5,-(a7)
                MOVE.W D0,D1
                LSL.W #2,D1
                ORI.W #8,D1
                MOVE.W D1,D2
                movem.l d0-d1/a0-a1/a5,-(a7)
                move.l a0,d1
                move.l a1,a5
                lea $dff000,a0
                bsr BlitWait
                bsr BlitterCode
                movem.l (a7)+,d0-d1/a0-a1/a5
                MOVE.L D0,D1
                MOVEA.L A1,A0
                BSR SetBorders
                ADDA.L D1,A0
                BSR SetBorders
                ADDA.L D1,A0
                BSR SetBorders
                movem.l (a7)+,d2/a5
                RTS

;A0 = $dff000
;D0 = Length of Source
;D1 = Source
;A5 = Destination

BlitterCode:
                bsr Modulu                  ;set Mode
                MOVE.L D1,$4C(A0)
```

```
                MOVE.L D1,$50(A0)
                MOVE.L A5,$54(A0)
                MOVE.W #$1DB1,$40(A0)
                MOVE.W #0,$42(A0)
                bsr StartBlit
                MOVE.L A5,$4C(A0)          ;Source B
                MOVE.L D1,$50(A0)          ;Source A
                MOVE.L A5,$54(A0)          ;Dest
                MOVE.W #$2d8c,$40(A0)
                bsr StartBlit
                movem.l d0-d1/a5,-(a7)
                ADD.L D0,D1
                SUBQ.L #2,D1
                ADDA.L D0,A5
                ADDA.L D0,A5
                SUBQ.L #2,A5
                MOVE.L D1,$4C(A0)
                MOVE.L D1,$50(A0)
                MOVE.L A5,$54(A0)
                MOVE.W #$DB1,$40(A0)
                MOVE.W #$1002,$42(A0)
                bsr StartBlit
                movem.l (a7)+,d0-d1/a5
                movem.l d0-d1/a5,-(a7)
                ADDA.L D0,A5
                MOVE.L A5,$4C(A0)
                MOVE.L D1,$50(A0)
                MOVE.L A5,$54(A0)
                MOVE.W #$1D8C,$40(A0)
                MOVE.W #0,$42(A0)
                bsr StartBlit
                movem.l (a7)+,d0-d1/a5
                rts

;start Blitter and wait for end of Blitter

StartBlit:
                MOVE.W d2,$dff058
BlitWait:       btst #14,$dff002
                bne.s BlitWait
                rts

;set Mode for coding
;>= A0 = $dff000
Modulu:
                movem.l d0/a1,-(a7)
                MOVEQ #0,D0
                LEA $44(A0),A1
                MOVE.L #-1,(A1)
                LEA $62(A0),A1
                MOVE.L D0,(A1)+
                MOVE.W D0,(A1)+
                ADDQ.L #8,A1
                MOVE.W #$5555,(A1)
                movem.l (a7)+,d0/a1
                rts
```

```
;erase Track with DOS
;>= A0 = Pointer to Track-Buffer
DOSClear:
                move.l #$444F5300,d0      ;DOS + 0
                move.w #$57f,d1
\DO1:           move.l d0,(a0)+           ;erase Track
                addq.b #1,d0
                dbf d1,\DO1
                rts

;shorten Track and store in memory
;>= A0 = Pointer to beginning of Track
;>= A1 = Pointer to destination address
;>= Length = number of Bytes to be shortened
;>= ShrtByte = Byte, which should be stored shorter
;=> A1 = Pointer to memory for shortened Block
Crunch:
                movem.l d2-d7/a2/a4,-(a7)
                clr.w d7
                clr.w d2
                move.w Length,d3      ;number of Bytes
                move.l MemoryLength,d6
\CHAnf:
                tst.w d7
                bne \CHEnd
                movea.l a0,a2     ;intermediate storage for Address
\CH4:           bsr \EraseBlock
                tst.w d0
                beq \CHAnf
                move.b (a0),d4
                cmp.b ShrtByte,d4
                beq \KF1                  ;Null byte
                cmp.b 1(a0),d4
                bne \CH10                 ;no sequel
                cmp.b 2(a0),d4
                beq \AF1                  ;other sequels
\CH10:          addq.l #1,a0
                subq.w #1,d3
                bne \CH4                  ;continue search
                bsr \NoResult
\CHEnd:
                move.b #$00,d0
                move.b d0,(a1)+
                subq.l #1,d6
                bcs CrunEnd
                move.l d6,MemoryLength
                move.l a1,MemoryBeg
                clr.l d0
                movem.l (a7)+,d2-d7/a2/a4
                rts
;short sequel

\KF1:           cmp.b 1(a0),d4
                bne \CH10                 ;no Null
                bsr \NoResult
```

```
                bsr CounterBytes
                cmp.w #$40,d1
                bcc \KF2                ;too large for a Byte
                ori.b #ShrtNull,d1
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd
                bra \KF4
\KF2:           cmp.w #$1000,d1
                bcc \KF3                ;too large, must be Word
                move.w d1,d0
                lsr.w #8,d0
                ori.b #MiddleNull,d0    ;Null sequence with Byte
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd
                bra \KF4
\KF3:           move.b #LongNull,d0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                move.w d1,d0
                lsr.w #8,d0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd
\KF4:           bra \CHAnf

;other sequence

\AF1:           bsr \NoResult
                bsr CounterBytes
                cmp.w #$40,d1
                bcc \AF2                ;too large for a Byte
                ori.b #ShrtNorm,d1
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd
                bra \AF4
\AF2:           cmp.w #$1000,d1
                bcc \AF3                ;too large, must be Word
                move.w d1,d0
                lsr.w #8,d0
                ori.b #MiddleNorm,d0    ;any sequence
                                        ;Byte
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                move.b d1,(a1)+
                subq.l #1,d6
```

```
                beq CrunEnd
                bra \AF4
\AF3:           move.b #LongNorm,d0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                move.w d1,d0
                lsr.w #8,d0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd
\AF4:           move.b d4,d0            ;store other Byte
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd
                bra \CHAnf

;found no sequence
\NoResult2:   move.w #1,Subtr           ;See CrunEnd2
                bra \KF20
\NoResult:    clr.w Subtr
\KF20:          move.l a0,a4
                sub.l a2,a4
                move.w a4,d1
                beq \KFEnd
                swap d1
                move.w a4,d1
                cmp.w #$40,d1
                bcc \CH5                ;too large for a Byte
                ori.b #ShrtNone,d1
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd2
                bra \CH6
\CH5:           cmp.w #$1000,d1
                bcc \CH7                ;too large, must be Word
                move.w d1,d0
                lsr.w #8,d0
                ori.b #MiddleNone,d0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd2
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd2
                bra \CH6
\CH7:           move.b #LongNone,D0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd2
                move.w d1,d0
                lsr.w #8,d0
                move.b d0,(a1)+
```

```
                subq.l #1,d6
                beq CrunEnd2
                move.b d1,(a1)+
                subq.l #1,d6
                beq CrunEnd2
\CH6:           swap d1
                andi.l #$ffff,d1
                sub.l d1,d6
                beq CrunEnd2
                bcs CrunEnd2
                bra \CH8
\CH9:           move.b (a2)+,(a1)+
\CH8:           dbf d1,\CH9
\KFEnd:         clr.w Subtr             ;See CrunEnd2
                rts

\EraseBlock:
                move.w d3,d0
                andi.w #$fe00,d0
                cmp.w d0,d3
                bne \LBEnd2
                move.l a0,a3
                move.w #$7e,d5
                move.l (a0)+,d4
                move.l d4,d0
                andi.l #$ff000000,d0
                cmp.l #$44000000,d0
                bne \LBNone
\LB1:           addq.b #1,d4
                cmp.l (a0)+,d4
                bne \LBNone
                dbf d5,\LB1
                bsr \NoResult2
                move.b #EmptyBlock,d0
                move.b d0,(a1)+
                subq.l #1,d6
                beq CrunEnd2
                clr.w d0                ;Ok-Message
                subi.w #$200,d3
                bne \LBEnd
                move.w #-1,d7           ;End mark
                bra \LBEnd
\LBNone:        move.l a3,a0
\LBEnd2:        move.w #-1,d0
\LBEnd:         rts

CounterBytes:     clr.w d1
\ZB2:           cmp.b (a0)+,d4
                bne \ZB1
                addq.w #1,d1
                subq.w #1,d3
                bne \ZB2
                move.w #-1,d7           ;End mark
\ZB1:
                subq.l #1,a0
                rts
```

```
CrunEnd2:       move.w Subtr,d2
\CE1:           adda.l #$4,a7
                dbf d2,\CE1
CrunEnd:
                movem.l (a7)+,d2-d7/a2/a4
                move.l #-1,d0
                rts

;bring Track to normal size
;>= A0 = Pointer to crunched Track
;>= A1 = Pointer to destination address

DeCrunch:
                move.b (a0)+,d0
                tst.b d0
                beq \DCEnd                  ;End mark
                move.b d0,d1
                andi.b #$c0,d1
                bne Shrt
                move.b d0,d1
                andi.b #$30,d1
                bne Middle
                move.b d0,d1
                cmpi.b #EmptyBlock,d1
                beq \BlockClear
                move.b (a0)+,d0
                lsl.w #8,d0
                move.b (a0)+,d0
                cmpi.b #LongNull,d1
                beq \NullLong
                cmpi.b #LongNorm,d1
                beq \NormLong
                cmpi.b #LongNone,d1
                beq \UndefLong
;error
                move.l $4,a6
                sub.l a5,a5
                move.l #$12345678,d7
                jsr -108(a6)
\DCEnd:         rts


;empty Block
\BlockClear:     move.w #$7f,d1
                move.l #$444F5300,d0
\LB1:           move.l d0,(a1)+
                addq.b #1,d0
                dbf d1,\LB1
                bra DeCrunch
;Long

\LA1:           move.b (a0)+,(a1)+
\UndefLong:     dbf d0,\LA1
```

```
                bra DeCrunch
\NullLong:      move.b Shrtbyte,d1
                bra \LA2
\NormLong:      move.b (a0)+,d1
                bra \LA2
\LA3:           move.b d1,(a1)+
\LA2:           dbf d0,\LA3
                bra DeCrunch
Shrt:
                cmpi.b #ShrtNull,d1
                beq \KU1
                cmpi.b #ShrtNorm,d1
                beq \KU2
;ShrtUndeff
                andi.w #$3f,d0
                bra \KU3
\KU4:           move.b (a0)+,(a1)+
\KU3:           dbf d0,\KU4
                bra DeCrunch
\KU1:           move.b Shrtbyte,d1
                bra \KU5
\KU2:           move.b (a0)+,d1
\KU5:           andi.w #$003f,d0
                bra \KU6
\KU7:           move.b d1,(a1)+
\KU6:           dbf d0,\Ku7
                bra DeCrunch
Middle:
                andi.w #$0f,d0
                lsl.w #8,d0
                move.b (a0)+,d0
                cmpi.b #MiddleNull,d1
                beq \KU1
                cmpi.b #MiddleNorm,d1
                beq \KU2
;MiddleUndeff
                bra \KU3
\KU4:           move.b (a0)+,(a1)+
\KU3:           dbf d0,\KU4
                bra DeCrunch
\KU1:           move.b Shrtbyte,d1
                bra \KU6
\KU2:           move.b (a0)+,d1
                bra \KU6
\KU7:           move.b d1,(a1)+
\KU6:           dbf d0,\Ku7
                bra DeCrunch

beg:
        move.w #$0008,$dff09a

        move.l gfxbase,a6
        lea    bitmap,a0
        move.b #1,d0
        move.w #320,d1
        move.w #200,d2          ;PAL uses 256
```

```
        jsr     InitBitMap(a6)
        move.l bit_adress,plane1
        move.l bit_adress,d1
        move.w d1,plane_lo
        swap    d1
        move.w d1,plane_hi

        lea     rastport,a1
        jsr     InitRastPort(a6)
        move.l #bitmap,r_bitmap

        move.l #ncopper,a0
        move.l cop_adress,a1
        move.l #copsize,d0
copy_loop:
        move.b (a0)+,(a1)+
        dbf     d0,copy_loop

        bsr   adresses
        move.l c1_adress,flash_adress

        move.l bit_adress,a0
        move.w #$27ff,d0
clear_loop:
        clr.b   (a0)+
        dbf     d0,clear_loop

        move.l gfxbase,a0
        move.w #$0080,$dff096
        move.l $6c,oldirq
        move.l #newirq,$6c
        move.l 50(a0),oldcopper
        move.l cop_adress,50(a0)
        move.w #$82b0,$dff096

new_start:
        move.w #7,x1
        move.w #54,y1
        move.b #$4f,lc
        move.b #$00,fc
        move.b #$03,tr
        move.b #$00,ws
        move.b #$01,vd
        move.b #$01,fa
        move.b #$00,dc1
        move.b #$00,dc2
        move.b #$01,sd
        move.b #$00,dd
        move.b #$01,sy
        move.b #$00,new
        move.b #$00,pointer1
        move.b #$00,color_ptr
        move.b drives,dd
        bsr   end_drive
        bsr   show_lc
        move.l #text1,text_ptr
```

```
        bsr  set_title
        bra  menu_control
exit:
        move.w #$8008,$dff09a
        move.l gfxbase,a0
        move.w #$0080,$dff096
        move.l oldirq,$6c
        move.l oldcopper,50(a0)
        move.w #$82b0,$dff096

no_DPuffer:
        move.l ExecBase,a6
        move.l cop_adress,a1
        move.l #copsize+2,d0
        jsr    FreeMem(a6)
no_copper:
        move.l ExecBase,a6
        move.l bit_adress,a1
        move.l #$2800,d0
        jsr    FreeMem(a6)
no_bitmap:
        move.l ExecBase,a6
        move.l gfxbase,a1
        jsr    CloseLibrary(a6)
no_gfxbase:
        clr.l  d0
        rts

newirq:
        move   SR,-(a7)
        movem.l a0-a6/d0-d7,-(a7)
        addq.b #1,waiting
        cmp.b  #2,waiting
        ble    endirq
        clr.b  waiting
        move.l flash_adress,a2
        cmpi.b #$00,color_ptr
        bne.s  irq_flash
        move.w #$00ee,(a2)
        bra.s  endirq
irq_flash:
        cmpi.b #$00,back
        beq.s  upward
downward:
        subi.w #$0011,(a2)
        cmpi.w #$0044,(a2)
        bcc.s  endirq
        move.b #$00,back
        bra.s  endirq
upward: addi.w #$0011,(a2)
        cmpi.w #$00ff,(a2)
        bcs.s  endirq
        move.b #$01,back
endirq: movem.l (a7)+,a0-a6/d0-d7
        move   (a7)+,SR
        dc.w $4ef9
```

```
oldirq: dc.l 0

menu_control:
        bsr  wait_key
        cmpi.b #$46,d0
        beq  destination_drive
        cmpi.b #$50,d0
        beq  start_copy
        cmpi.b #$51,d0
        beq  first_cylinder
        cmpi.b #$52,d0
        beq  last_cylinder
        cmpi.b #$53,d0
        beq  how_many_tries
        cmpi.b #$54,d0
        beq  write_serveral_times
        cmpi.b #$55,d0
        beq  verify_destination
        cmpi.b #$56,d0
        beq  fast_copy
        cmpi.b #$57,d0
        beq  deepcopy_1
        cmpi.b #$58,d0
        beq  deepcopy_2
        cmpi.b #$59,d0
        beq  source_drive
        cmpi.b #$21,d0
        beq  synccorrection
        bra.s  menu_control
start_copy:
        move.l c2_adress,flash_adress
        move.b #$01,color_ptr
        bsr  show_start
start_copy2:
        bsr  wait_key
        cmpi.b #$45,d0
        beq  new_start
        cmpi.b #$44,d0                ;Return
        beq.s  end_start
        cmpi.b #$43,d0
        bne.s  start_copy2
end_start:
        move.b #$00,color_ptr
        move.w #28,yl
        bsr    cl2
        move.w #10,d0
estlop: bsr    Timer
        dbf    d0,estlop
        bsr clear_eol
        bsr copy_start                ;Copy-Routine
        cmp.w #Escape,ErrorFlag
        beq new_start
        cmp.w #diskprotect,errorflag
        beq start_copy
        cmp.w #NoDisk,errorflag
        beq start_copy
```

```
        cmp.w #NotProtect,ErrorFlag
        beq.s start_copy
        bsr  cl1
esd1:   bsr  wait_key
        cmpi.b #$40,d0
        bne.s  esd1
esd2:   bra new_start

destination_drive:
        move.l cl3_adress,flash_adress
        move.b #$01,color_ptr
destination1:
        bsr  wait_key
        cmpi.b #$0a,d0
        beq.s  d_drive0
        cmpi.b #$01,d0
        beq.s  d_drive1
        cmpi.b #$02,d0
        beq  d_drive2
        cmpi.b #$03,d0
        beq  d_drive3
        cmpi.b #$43,d0
        beq.s  end_destination
        cmpi.b #$44,d0
        bne.s  destination1
end_destination:
        cmpi.b #$00,dd
        beq.s  destination1
        move.b #$00,color_ptr
        bra  menu_control
d_drive0:
        btst   #0,drives
        beq  end_drive
        btst   #0,dd
        beq.s  dd0_0
        bclr   #0,dd
        bra.s  dd0_1
dd0_0:  bset   #0,dd
        move.b #1,drv
dd0_1:  bra  end_drive

d_drive1:
        btst   #1,drives
        beq  end_drive
        btst   #1,dd
        beq.s  dd1_0
        bclr   #1,dd
        bra.s  dd1_1
dd1_0:  bset   #1,dd
        move.b #2,drv
dd1_1:  bra.s  end_drive

d_drive2:
        btst   #2,drives
        beq.s  end_drive
        btst   #2,dd
```

```
        beq.s  dd2_0
        bclr   #2,dd
        bra.s  dd2_1
dd2_0:  bset   #2,dd
        move.b #4,drv
dd2_1:  bra.s  end_drive

d_drive3:
        btst   #3,drives
        beq.s  end_drive
        btst   #3,dd
        beq.s  dd3_0
        bclr   #3,dd
        bra.s  end_drive
dd3_0:  bset   #3,dd
        move.b #8,drv
end_drive:
        cmpi.b #$01,dc1
        bne.s  st1
        bra.s  st2
st1:    cmpi.b #$01,dc2
        bne  steve1
st2:    move.b drv,dd
        bra  steve
steve1: btst   #0,sd
        beq.s  ste1
        andi.b #$0e,dd
        cmpi.b #$00,dd
        bne.s  steve
        bset   #0,dd
        bra.s  steve
ste1:   btst   #1,sd
        beq.s  ste2
        andi.b #$0d,dd
        cmpi.b #$00,dd
        bne.s  steve
        bset   #1,dd
        bra.s  steve
ste2:   btst   #2,sd
        beq.s  ste3
        andi.b #$0b,dd
        cmpi.b #$00,dd
        bne.s  steve
        bset   #2,dd
        bra.s  steve
ste3:   btst   #3,sd
        beq.s  steve
        andi.b #$07,dd
        cmpi.b #$00,dd
        bne.s  steve
        bset   #3,dd
steve:  btst   #0,dd
        bne.s  end_dd1
        lea    off_text,a0
        bra.s  end_dd2
end_dd1:lea    on_text2,a0
```

```
end_dd2:bsr  set_drive0
        btst   #1,dd
        bne.s  end_dd3
        lea    off_text,a0
        bra.s  end_dd4
end_dd3:lea    on_text2,a0
end_dd4:bsr  set_drive1
        btst   #2,dd
        bne.s  end_dd5
        lea    off_text,a0
        bra.s  end_dd6
end_dd5:lea    on_text2,a0
end_dd6:bsr  set_drive2
        btst   #3,dd
        bne.s  end_dd7
        lea    off_text,a0
        bra.s  end_dd8
end_dd7:lea    on_text2,a0
end_dd8:bsr  set_drive3
        cmpi.b #$00,pointer1
        bne.s  end_dd9
        move.b #$01,pointer1
        rts
end_dd9:bra  destination1

set_drive0:
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #61,d0
        move.w #190,d1          ;PAL 213
        jsr    Movee(a6)
        bra.s  set_text
set_drive1:
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #126,d0
        move.w #190,d1          ;PAL 213
        jsr    Movee(a6)
        bra.s  set_text
set_drive2:
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #190,d0
        move.w #190,d1          ;PAL 213
        jsr    Movee(a6)
        bra.s  set_text
set_drive3:
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #254,d0
        move.w #190,d1          ;PAL 213
        jsr    Movee(a6)
set_text:
        lea    rastport,a1
        move.w #$0003,d0
        jsr    Textout(a6)
```

```
        rts

first_cylinder:
        move.l c3_adress,flash_adress
        move.b #$01,color_ptr
        bsr  wait_key
        cmpi.b #$4f,d0
        beq.s  fc_down
        cmpi.b #$4e,d0
        beq.s  fc_up
        cmpi.b #$43,d0
        beq  compare_fc
        cmpi.b #$44,d0
        bne.s  first_cylinder
compare_fc:
        move.b lc,d0
        cmp.b fc,d0
        beq  compare_fc2
        blt.s  first_cylinder
compare_fc2:
        move.b #$00,color_ptr
        bra  menu_control
fc_down:subi.b #$01,fc
        cmpi.b #$ff,fc
        bne.s  fc_down2
        move.b #$00,fc
fc_down2:
        bra  show_fc
fc_up:  addi.b #$01,fc
        cmpi.b #$52,fc
        bne.s  fc_down2
        move.b #$51,fc
show_fc:lea    fc_text,a0
        move.b fc,d0
        bsr  byte_calculate
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #295,d0
        move.w #64,d1
        jsr    Movee(a6)
        lea    rastport,a1
        lea    fc_text,a0
        move.w #$0002,d0
        jsr    Textout(a6)
        bra  first_cylinder

last_cylinder:
        move.l c4_adress,flash_adress
        move.b #$01,color_ptr
        bsr  wait_key
        cmpi.b #$4f,d0
        beq.s  lc_down
        cmpi.b #$4e,d0
        beq.s  lc_up
        cmpi.b #$43,d0
        beq  compare_lc
```

```
        cmpi.b #$44,d0
        bne.s  last_cylinder
compare_lc:
        move.b fc,d0
        cmp.b lc,d0
        beq  compare_lc2
        bge.s  last_cylinder
compare_lc2:
        move.b #$00,color_ptr
        bra  menu_control
lc_down:subi.b #$01,lc
        cmpi.b #$ff,lc
        bne.s  lc_down2
        move.b #$00,lc
lc_down2:
        bra  lcc
lc_up:  addi.b #$01,lc
        cmpi.b #$52,lc
        bne.s  lc_down2
        move.b #$51,lc
lcc:    bsr.s  show_lc
        bra  last_cylinder
show_lc:lea    lc_text,a0
        move.b lc,d0
        bsr  byte_calculate
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #295,d0
        move.w #74,d1
        jsr    Movee(a6)
        lea    rastport,a1
        lea    lc_text,a0
        move.w #$0002,d0
        jsr    Textout(a6)
        rts

how_many_tries:
        move.l c5_adress,flash_adress
        move.b #$01,color_ptr
tries1: lea    tr_text,a0
        bsr  wait_key
        cmpi.b #$4e,d0
        beq.s  tries_up
        cmpi.b #$4f,d0
        beq.s  tries_down
        cmpi.b #$43,d0
        beq.s  end_tries
        cmpi.b #$44,d0
        bne.s  tries1
end_tries:
        move.b #$00,color_ptr
        bra  menu_control
tries_up:
        addi.b #$01,tr
        cmpi.b #$0a,tr
        bne.s  tries_up2
```

```
        move.b #$09,tr
tries_up2:
        move.b tr,d0
        addi.w #$30,d0
        move.b d0,(a0)
        bra.s  tries2
tries_down:
        subi.b #$01,tr
        cmpi.b #$00,tr
        bne.s  tries_down2
        move.b #$01,tr
tries_down2:
        move.b tr,d0
        addi.w #$30,d0
        move.b d0,(a0)
tries2: move.l gfxbase,a6
        lea    rastport,a1
        move.w #303,d0
        move.w #84,d1
        jsr    Movee(a6)
        lea    rastport,a1
        lea    tr_text,a0
        move.w #$0001,d0
        jsr    Textout(a6)
        bra  tries1

synccorrection:
        move.l c12_adress,flash_adress
        move.b #$01,color_ptr
        cmpi.b #$01,sy
        bne.s  sync2
        clr.b  sy
        lea    off_text,a0
        bra.s  sync3
sync2:  move.b #$01,sy
        lea    on_text,a0
sync3:  lea    rastport,a1
        move.w #287,d0
        move.w #157,d1
        jsr    Movee(a6)
        lea    rastport,a1
        move.w #3,d0
        jsr    Textout(a6)
        move.b #$00,color_ptr
        bra  menu_control

write_serveral_times:
        move.l c6_adress,flash_adress
        move.b #$01,color_ptr
        move.b sd,d0
        cmp.b dd,d0
        bne  wait_return
        cmpi.b #$00,ws
        bne.s  write_s2
write_s1:
        move.b #$01,ws
```

```
        lea     on_text,a0
        bra.s   end_write_serveral
write_s2:
        move.b #$00,ws
        lea     off_text,a0
end_write_serveral:
        move.l gfxbase,a6
        lea     rastport,a1
        move.w #287,d0
        move.w #94,d1
        jsr     Movee(a6)
        lea     rastport,a1
        move.w #$0003,d0
        jsr     Textout(a6)
esw:    move.b #$00,color_ptr
        bra  menu_control

verify_destination:
        move.b dc1,d0
        or.b dc2,d0
        tst.b d0
        bne menu_control
        move.l c7_adress,flash_adress
        move.b #$01,color_ptr
        cmpi.b #$00,vd
        bne.s   verify_d2
verify_d1:
        move.b #$01,vd
        lea     on_text,a0
        bra.s   end_verify
verify_d2:
        move.b #$00,vd
        lea     off_text,a0
end_verify:
        bsr verify_d3
        bra  menu_control
verify_off:
        move.b #$00,vd
        lea     off_text,a0
verify_d3:
        move.l gfxbase,a6
        lea     rastport,a1
        move.w #287,d0
        move.w #104,d1
        jsr     Movee(a6)
        lea     rastport,a1
        move.w #$0003,d0
        jsr     Textout(a6)
        move.b #$00,color_ptr
        rts

fast_copy:
        move.l c8_adress,flash_adress
        move.b #$01,color_ptr
        cmpi.b #$01,dd
        beq.s   fcop1
```

```
        cmpi.b #$02,dd
        beq.s  fcop1
        cmpi.b #$04,dd
        beq.s  fcop1
        cmpi.b #$08,dd
        beq.s  fcop1
        bra  wait_return
fcop1:  cmpi.b #$01,fa
        bne.s  fastcopy1
        move.b #$00,fa
        move.b #$01,dc1
        move.b #$00,dc2
        bra.s  end_fast
fastcopy1:
        move.b #$01,fa
        move.b #$00,dc1
        move.b #$00,dc2
end_fast:
        cmp.b #ON,DC1
        bne fcop2
        bsr verify_off
fcop2:  bsr  show_copies
        move.b #$00,color_ptr
        bra  menu_control

deepcopy_1:
        bsr verify_off
        move.l c9_adress,flash_adress
        move.b #$01,color_ptr
        cmpi.b #$01,dd
        beq.s  deep4
        cmpi.b #$02,dd
        beq.s  deep4
        cmpi.b #$04,dd
        beq.s  deep4
        cmpi.b #$08,dd
        beq.s  deep4
        bra.s  wait_return
deep4:  cmpi.b #$00,dc1
        beq.s  deep1
        move.b #$00,fa
        move.b #$00,dc1
        move.b #$01,dc2
        bra.s  end_deep1
deep1:  move.b #$00,fa
        move.b #$01,dc1
        move.b #$00,dc2
end_deep1:
        bra  end_fast

wait_return:
        move.l #text12,text_ptr
        move.w #16,x1
        move.w #28,y1
        bsr    set_text3
wait_r3:bsr    wait_key
```

```
        cmpi.b #$44,d0
        beq.s  wait_r2
        cmpi.b #$43,d0
        bne.s  wait_return
wait_r2:
        bsr    BegText
        bra    end_fast
wait_r4:bsr    wait_key
        cmpi.b #$45,d0
        beq.s  wait_r5
        cmpi.b #$44,d0
        beq    \W1
        cmpi.b #$43,d0
        beq    \W1
        cmpi.b #$40,d0
        bne.s  wait_r4
\W1:    move.l d0,-(a7)
        bsr    clear_eol
        move.l (a7)+,d0
        clr.l  d0
        rts
wait_r5:move.l #-1,d0
        move.w #Escape,ErrorFlag
        rts

deepcopy_2:
        bsr    verify_off
        move.l c10_adress,flash_adress
        move.b #$01,color_ptr
        cmpi.b #$01,dd
        beq.s  deep3
        cmpi.b #$02,dd
        beq.s  deep3
        cmpi.b #$04,dd
        beq.s  deep3
        cmpi.b #$08,dd
        beq.s  deep3
        bra    wait_return
deep3:  cmpi.b #$01,dc2
        beq.s  deep2
        move.b #$00,fa
        move.b #$00,dc1
        move.b #$01,dc2
        bra.s  end_deep2
deep2:  move.b #$01,fa
        move.b #$00,dc1
        move.b #$00,dc2
end_deep2:
        bra    end_fast
show_copies:
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #287,d0
        move.w #114,d1
        jsr    Movee(a6)
        lea    rastport,a1
```

```
        cmpi.b #$00,fa
        beq.s  show1
        lea    on_text,a0
        bra.s  show2
show1:  lea    off_text,a0
show2:  move.w #$0003,d0
        jsr    Textout(a6)
        lea    rastport,a1
        move.w #287,d0
        move.w #124,d1
        jsr    Movee(a6)
        lea    rastport,a1
        cmpi.b #$00,dc1
        beq.s  show3
        lea    on_text,a0
        bra.s  show4
show3:  lea    off_text,a0
show4:  move.w #$0003,d0
        jsr    Textout(a6)
        lea    rastport,a1
        move.w #287,d0
        move.w #134,d1
        jsr    Movee(a6)
        lea    rastport,a1
        cmpi.b #$00,dc2
        beq.s  show5
        lea    on_text,a0
        bra.s  show6
show5:  lea    off_text,a0
show6:  move.w #$0003,d0
        jsr    Textout(a6)
        rts

source_drive:
        move.l cl1_adress,flash_adress
        move.b #$01,color_ptr
source1:
        bsr    wait_key
        cmpi.b #$0a,d0
        beq.s  s_drive0
        cmpi.b #$01,d0
        beq.s  s_drive1
        cmpi.b #$02,d0
        beq    s_drive2
        cmpi.b #$03,d0
        beq    s_drive3
        cmpi.b #$43,d0
        beq.s  source2
        cmpi.b #$44,d0
        bne.s  source1
source2:move.b pointer1,d7
        clr.b  pointer1
        bsr    end_drive
        move.b d7,pointer1
        move.b #$00,color_ptr
        bra    menu_control
```

```
s_drive0:
        btst    #$00,drives
        beq     source1
s_dd0:  move.b #$01,sd
        move.b #$00,d0
        bra     set_s_drive
s_drive1:
        btst    #$01,drives
        beq     source1
s_dd1:  move.b #$02,sd
        move.b #$01,d0
        bra     set_s_drive
s_drive2:
        btst    #$02,drives
        beq     source1
s_dd2:  move.b #$04,sd
        move.b #$02,d0
        bra.s   set_s_drive
s_drive3:
        btst    #$03,drives
        beq     source1
        move.b #$08,sd
s_dd3:  move.b #$03,d0
set_s_drive:
        lea     sd_text,a0
        addi.b #$30,d0
        move.b d0,(a0)
        move.l gfxbase,a6
        lea     rastport,a1
        move.w #303,d0
        move.w #144,d1
        jsr     movee(a6)
        lea     rastport,a1
        move.w #$0001,d0
        jsr     Textout(a6)
        bra     source1
byte_calculate:
        cmpi.w #$0a,d0
        bmi.s   byte2
        divu    #$0a,d0
        move.w d0,d1
        addi.w #$30,d1
        swap    d0
        bra.s   byte3
byte2:  move.w #$0030,d1
byte3:  move.b d1,(a0)+
        addi.w #$0030,d0
        move.b d0,(a0)+
        rts

adresses:
        move.l cop_adress,d0
        move.l #color1,d1
        move.l #ncopper,d2
        sub.l   d2,d1
        add.l   d1,d0
```

```
        move.l d0,c1_adress
        addi.l #$000010,d0
        move.l d0,c2_adress
        addi.l #$000008,d0
        move.l d0,c3_adress
        addi.l #$000008,d0
        move.l d0,c4_adress
        addi.l #$000008,d0
        move.l d0,c5_adress
        addi.l #$000008,d0
        move.l d0,c6_adress
        addi.l #$000008,d0
        move.l d0,c7_adress
        addi.l #$000008,d0
        move.l d0,c8_adress
        addi.l #$000008,d0
        move.l d0,c9_adress
        addi.l #$000008,d0
        move.l d0,c10_adress
        addi.l #$000008,d0
        move.l d0,c11_adress
        addi.l #$000008,d0
        move.l d0,c12_adress
        addi.l #$000010,d0
        move.l d0,c13_adress
        rts

set_title:
        clr.l  d6
        move.w #$0009,d6
set_text1:
        move.l gfxbase,a6
        bsr    set_text2
        addi.l #$000026,text_ptr
        addi.w #$000a,y1
        dbf    d6,set_text1
        lea    floppy,a0
        move.l bit_adress,a1
        add.l  #$001c26,a1
        move.l a1,a2
        bsr.s  box
        lea    floppy,a0
        move.l a1,a2
        add.l  #$000008,a2
        bsr.s  box
        lea    floppy,a0
        move.l a1,a2
        add.l  #$000010,a2
        bsr.s  box
        lea    floppy,a0
        move.l a1,a2
        add.l  #$000018,a2
        bsr.s  box
        bra.s  set_work
box:    rts                         ;Remove this rts for PAL systems
        move.w #floppy_s,d0         ;draws graphic disk drive onscreen
```

```
bit_copy_loop:
        move.b (a0)+,(a2)+
        move.b (a0)+,(a2)+
        move.b (a0)+,(a2)+
        move.b (a0)+,(a2)+
        move.b (a0)+,(a2)+
        move.b (a0)+,(a2)+
        add.l  #$22,a2
        dbf    d0,bit_copy_loop
        rts
set_work:
        move.w #44,y2
        bsr    draw_line
        move.w #148,y2
        bsr    draw_line
        move.w #218,y2
        bsr    draw_line
        move.w #250,y2
        bsr    draw_line
        move.w #160,y2
        bsr    draw_line
        move.w #157,y1
        move.l #text11,text_ptr
        bsr    set_text2
        move.w #170,y1
        move.w #$0001,d2
        move.l #text2,text_ptr
set_loop2:
        bsr    set_text2
        addi.l #$000026,text_ptr
        addi.w #$000a,y1
        dbf    d2,set_loop2
        move.w #$0001,d2
        move.l #text3,text_ptr
        move.w #200,y1          ;PAL uses 230
        bsr    set_text2;       ;added to NTSC
        move.l #text3a,text_ptr ;added to NTSC
        move.w #7,x1            ;to position text
        move.w #210,y1          ;PAL uses 245
set_loop3:
        bsr    set_text2
        addi.w #$000f,y1
        addi.l #$000026,text_ptr
        dbf    d2,set_loop3
        lea    rastport,a1
        move.b #1,d0
        jsr    SetAPen(a6)
        lea    rastport,a1
        move.w #10,d0
        move.w #10,d1
        move.w #309,d2
        move.w #40,d3
        jsr    RectFill(a6)
        lea    rastport,a1
        move.b #0,d0
        jsr    SetAPen(a6)
```

```
        lea    rastport,a1
        move.w #11,d0
        move.w #11,d1
        move.w #308,d2
        move.w #39,d3
        jsr    RectFill(a6)
        lea    rastport,a1
        move.b #1,d0
        jsr    SetAPen(a6)
BegText: move.l #text4,text_ptr
        move.w #16,x1
        move.w #18,y1
        bsr    set_text3
        move.l #text20,text_ptr
        move.w #28,y1
        bsr    set_text3
        move.l #text21,text_ptr
        move.w #38,y1
        bsr    set_text3
        rts

draw_line:
        lea    rastport,a1
        move.w #$0000,d0
        move.w y2,d1
        jsr    Movee(a6)
        lea    rastport,a1
        move.w #$013f,d0
        move.w y2,d1
        jsr    Draw(a6)
        rts

set_text2:
        move.l a6,-(a7)
        move.l gfxbase,a6
        lea    rastport,a1
        move.w x1,d0
        move.w y1,d1
        jsr    Movee(a6)
        lea    rastport,a1
        move.l text_ptr,a0
        move.w #$0026,d0
        jsr    Textout(a6)
        move.l (a7)+,a6
        rts

set_text3:
        move.l a6,-(a7)
        move.l gfxbase,a6
        lea    rastport,a1
        move.w x1,d0
        move.w y1,d1
        jsr    Movee(a6)
        lea    rastport,a1
        move.l text_ptr,a0
        move.w #$0024,d0
```

```
        jsr     Textout(a6)
        move.l  (a7)+,a6
        rts

show_start:
        lea     text6,a0
        btst    #0,dd
        bne     show_s1
        btst    #0,sd
        beq.s   show_start1
show_s1:lea     df0,a1
        bsr     copy_drives
show_start1:
        btst    #1,dd
        bne.s   show_s2
        btst    #1,sd
        beq.s   show_start2
show_s2:lea     df1,a1
        bsr     copy_drives
show_start2:
        btst    #2,dd
        bne.s   show_s3
        btst    #2,sd
        beq.s   show_start3
show_s3:lea     df2,a1
        bsr.s   copy_drives
show_start3:
        btst    #3,dd
        bne.s   show_s4
        btst    #3,sd
        beq.s   show_start4
show_s4:lea     df3,a1
        bsr.s   copy_drives
show_start4:
        move.w #16,x1
        move.w #18,y1
        move.w #$002,d2
        move.l #text5,text_ptr
show_loop:
        bsr     set_text3
        addi.w #$000a,y1
        addi.l #$000024,text_ptr
        dbf     d2,show_loop
        move.w #16,d0
        lea     text6,a0
lop:    move.b #$20,(a0)+
        dbf     d0,lop
        rts

copy_drives:
        move.w #$0003,d0
copy_d_loop:
        move.b (a1)+,(a0)+
        dbf     d0,copy_d_loop
        rts
```

```
read_error:
        clr.l   d0
        move.b  cylinder,d0
        lea     text7,a0
        add.l   #$00001a,a0
        bsr     byte_calculate
        move.b  side,d0
        add.l   #$000006,a0
        bsr     byte_calculate
        move.w  #7,x1
        move.w  #200,y1          ;PAL uses 230
        move.l  #text7,text_ptr
        bra     set_text2
write_error:
        clr.l   d0
        move.b  cylinder,d0
        lea     text8,a0
        add.l   #$00001b,a0
        bsr     byte_calculate
        move.b  side,d0
        add.l   #$000006,a0
        bsr     byte_calculate
        move.w  #7,x1
        move.w  #200,y1          ;PAL uses 230
        move.l  #text8,text_ptr
        bra     set_text2
clear_error:
        move.l  #text3,text_ptr
        move.w  #7,x1
        move.w  #200,y1       ;PAL uses 230
        bra     set_text2
         move.l #text3a,text_ptr ;added to NTSC
        move.w  #7,x1             ;to position text
        move.w  #210,y1           ;(PAL uses 245)
        bra     set_text2           ;on the screen

reading_cyl:
        clr.l   d0
        move.b  cylinder,d0
        lea     rcyl_text,a0
        bsr     byte_calculate
        lea     rcyl_text,a0
        move.l  gfxbase,a6
        lea     rastport,a1
        move.w  #127,d0
        move.w  #210,d1          ;PALS uses 245
        jsr     Movee(a6)
        lea     rastport,a1
        move.w  #$0002,d0
        jsr     Textout(a6)
        rts
writing_cyl:
        clr.l   d0
        move.b  cylinder,d0
        lea     wcyl_text,a0
        bsr     byte_calculate
```

```
        lea    wcyl_text,a0
        move.l gfxbase,a6
        lea    rastport,a1
        move.w #271,d0
        move.w #210,d1          ;PAL uses 245
        jsr    Movee(a6)
        lea    rastport,a1
        move.w #$0002,d0
        jsr    Textout(a6)
        rts

insert_source:
        move.l #text9,text_ptr
        bra.s  melvin
insert_destination:
        move.l #text10,text_ptr
        bra.s  melvin
protect_source:
        bsr  cl
        move.l #text13,text_ptr
        bra.s  melvin
protect_destination:
        bsr  cl
        move.l #text14,text_ptr
melvin:
        move.w #16,x1
        move.w #28,y1
        bsr    set_text3
        move.l #text5,text_ptr
        move.w #18,y1
        bsr    set_text3
        bra    wait_r4
clear_eol:
        move.l #text15,text_ptr
        move.w #16,x1
        move.w #28,y1
        bsr    set_text3
        move.w #18,y1
        bra    set_text3

write_b_again:          ;Check write again Y/N
        bsr  cl
        move.l #text17,text_ptr
        move.w #16,x1
        move.w #28,y1
        bsr    set_text3
wba1:   bsr    wait_key
        cmpi.b #$45,d0   ;check esc
        beq.s  wba4
        cmpi.b #$15,d0   ;German keyboards use 31 y and z reversed
        beq.s  wba2
        cmpi.b #$36,d0   ;check n
        bne.s  wba1
        move.l #-1,d0
        bra.s  wba3
wba2:   clr.l  d0
```

```
wba3:   move.l d0,-(a7)
        bsr    clear_eol
        move.l (a7)+,d0
        rts
wba4:   move.w #escape,errorflag
        rts

compare_drives:
        move.w #18,y1
        bsr    cl2
        lea    text19,a0
        clr.w  d2
cdl1:   lea    df0,a1
        btst   d2,vererrflag
        beq.s  cdl4
        move.w d2,d0
        lsl    #2,d0
        add.l  d0,a1
        bsr.s  cdl2
cdl4:   addq.w #1,d2
        cmpi.w #4,d2
        bne.s  cdl1
        move.l #text18,text_ptr
        move.w #28,y1
        bra    set_text3
cdl2:   move.w #3,d1
cdl3:   move.b (a1)+,(a0)+
        dbf    d1,cdl3
        rts

wait_key:
        movem.l d1-d7,-(a7)
        move.w Keybrdclr,d1
l8:     move.b IntCon,d0
        btst   #7,d1
        bne    l1
        sub.l  #1,Keybrdcnt
        beq    Keybrepeat
l1:     btst   #3,d0
        beq    l8
        move.b Key,d0
        ori.b  #$40,Cont
        not.b  d0
        ror.b  #1,d0
        move.w #$600,d1
l5:     dbf    d1,l5
        andi.b #$bf,Cont
        move.l #MaxWait,Cntwait
        move.l #MaxWait,Keybrdcnt
        move.w d0,Keybrdclr
Keybtdend:
        movem.l (a7)+,d1-d7
        rts
Keybrepeat:
        move.l Cntwait,d1
        cmp.l  #MinWait,d1
```

```
        bcs.s  repeat1
        sub.l  #$800,d1
        move.l d1,Cntwait
repeat1:move.l d1,Keybrdcnt
        move.w Keybrdclr,d0
        bra.s  Keybtdend

get_key:
        move.w $dff01e,d1
        btst   #3,d1
        beq    \l1
        move.b Key,d0
        ori.b  #$40,Cont
        not.b  d0
        ror.b  #1,d0
        move.w #$600,d1
\l2:    dbf    d1,\l2
        andi.b #$bf,Cont
\l1:    rts

cl:     move.w #28,y1
cl2:    move.w #16,x1
        move.l #text15,text_ptr
        bsr    set_text3
        addi.w #10,y1
        bsr    set_text3
        rts

cl1:    bsr.s  cl
        move.w #18,y1
        bsr    set_text3
        move.w #28,y1
        move.l #text16,text_ptr
        bsr    set_text3
        rts


gfxname:
         dc.b "graphics.library",0
         align.w
gfxbase:        dc.l 0
oldcopper:      dc.l 0
bit_adress:     dc.l 0
cop_adress:     dc.l 0
c1_adress:      dc.l 0
c2_adress:      dc.l 0
c3_adress:      dc.l 0
c4_adress:      dc.l 0
c5_adress:      dc.l 0
c6_adress:      dc.l 0
c7_adress:      dc.l 0
c8_adress:      dc.l 0
c9_adress:      dc.l 0
c10_adress:     dc.l 0
c11_adress:     dc.l 0
c12_adress:     dc.l 0
```

```
cl3_adress:     dc.l 0

rastport:       blk.l  1,0
r_bitmap:       blk.l 24,0

bitmap:         blk.l  2,0
plane1:         blk.l  8,0

ncopper:
                dc.w $008e,$2681,$0090,$24c1
                dc.w $0092,$0038,$0094,$00d0
                dc.w $00e0
plane_hi:       dc.w $0000,$00e2
plane_lo:       dc.w $0000
                dc.w $0100,$1200,$0102,$0000,$0104,$0000
                dc.w $0108,$0000,$010a,$0000
                dc.w $0120,$0000,$0122,$0000
                dc.w $0180,$0000,$0182
color1:         dc.w $00ee,$0d01,$fffe,$0182
                dc.w $00ee,$5401,$fffe,$0182
color2:         dc.w $00ee,$6001,$fffe,$0182
color3:         dc.w $00ee,$6a01,$fffe,$0182
color4:         dc.w $00ee,$7201,$fffe,$0182
color5:         dc.w $00ee,$7d01,$fffe,$0182
color6:         dc.w $00ee,$8601,$fffe,$0182
color7:         dc.w $00ee,$9001,$fffe,$0182
color8:         dc.w $00ee,$9a01,$fffe,$0182
color9:         dc.w $00ee,$a401,$fffe,$0182
color10:        dc.w $00ee,$af01,$fffe,$0182
color11:        dc.w $00ee,$b701,$fffe,$0182
color12:        dc.w $00ee,$c501,$fffe,$0182
                dc.w $00ee,$c701,$fffe,$0182
color13:        dc.w $00ee,$ff01,$fffe,$0182
                dc.w $00ee,$ffff,$fffe
                dc.w $0000,$0000
copend:
copsize = copend - ncopper

drives:         dc.b 0
d_drives:       dc.b 0
s_drives:       dc.b 0
color_ptr:      dc.b 0
back:           dc.b 0
pointer1:       dc.b 0
cylinder:       dc.b 0
side:           dc.b 0
new:            dc.b 0
drv:            dc.b 0
        align.w
flash_adress:   dc.l 0
text_ptr:       dc.l 0

x1:             dc.w 7
y1:             dc.w 54
y2:             dc.w 0
```

```
keybrdclr:      dc.w $ffff
cntwait:        dc.l MaxWait
keybrdcnt:      dc.l MaxWait

fc_text:        dc.b "00"
ralf:           dc.b 0
lc_text:        dc.b "79"
tr_text:        dc.b "0"
sd_text:        dc.b "0"
rcyl_text:      dc.b "00"
wcyl_text:      dc.b "00"
on_text:        dc.b ".ON"
on_text2:       dc.b " ON"
off_text:       dc.b "OFF"
df0:            dc.b "DF0 "
df1:            dc.b "DF1 "
df2:            dc.b "DF2 "
df3:            dc.b "DF3 "

text1:  dc.b "F1  = START COPY.....................""
        dc.b "F2  = FIRST CYLINDER (CRSR).........00"
        dc.b "F3  = LAST  CYLINDER (CRSR).........79"
        dc.b "F4  = HOW MANY TRIES (CRSR)..........3"
        dc.b "F5  = WRITE SEVERAL TIMES..........OFF"
        dc.b "F6  = VERIFY DESTINATION...........ON"
        dc.b "F7  = FASTCOPY.....................ON"
        dc.b "F8  = DEEPCOPY 1..................OFF"
        dc.b "F9  = DEEPCOPY 2..................OFF"
        dc.b "F10 = SOURCE DRIVE (0/1/2/3).......DF0"
text2:  dc.b "DEL = DESTINATION DRIVE (0/1/2/3).....""
        dc.b "      DF0:    DF1:    DF2:    DF3:    "
text3:  dc.b "         STATUS: 00, OK,00,00         "
text3a: dc.b "  READING CYL. 00 / WRITING CYL. 00   "
;text3a added for text positioning on NTSC systems
text4:  dc.b "       A M I G A - COPY  V1.2         "
text5:  dc.b "        (ESC) TERMINATES COPY         "
        dc.b "INSERT DISK(S) IN  "
text6:  dc.b "                   "
        dc.b "PRESS RETURN OR ENTER WHEN READY !°!"
text7:  dc.b "STATUS: READ-ERROR ON CYL.00 SIDE 00  "
text8:  dc.b "STATUS: WRITE-ERROR ON CYL.00 SIDE 00 "
text9:  dc.b "    PLEASE INSERT SOURCE DISK !!      "
text10: dc.b "  PLEASE INSERT DESTINATION DISK !    "
text11: dc.b " S  = SYNCCORRECTION................ON"
text12: dc.b "       ONLY ONE DESTINATION !!        "
text13: dc.b " SOURCE DISK INSN'T WRITEPROTECTED.   "
text14: dc.b "DESTINATION DISK IS WRITEPROTECTED.   "
text15: dc.b "                                      "
text16: dc.b "          COPY COMPLETE !!            "
text17: dc.b "    WRITE BUFFER AGAIN ??? (Y/N)      "
text18: dc.b " ERROR ON DRIVE(S) "
text19: dc.b "                   "
text20: dc.b "            WRITTEN BY :              "
text21: dc.b "   R. GELFAND AND S. THUBEAUVILLE     "
       align.w
floppy:
```

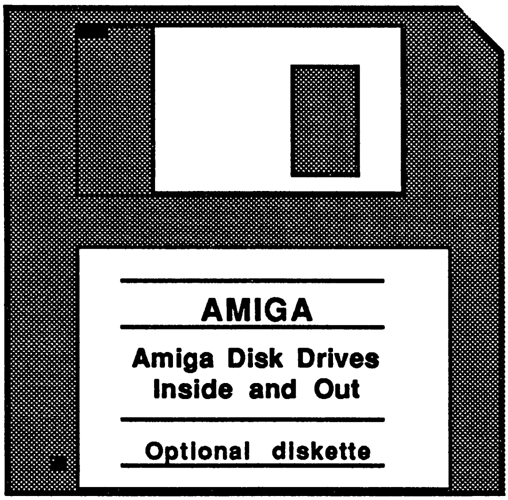
```
 dc.l $00000000,$00000000,$00000000,$00000000,$000007ff,$ffffffe0
 dc.l $04000000,$002005ff,$ffffffa0,$05ffffff,$ffa00400,$00000020
 dc.l $04ffffff,$ff200880,$00000110,$08800000,$01100880,$1ff80110
 dc.l $08801008,$011008bf,$f00ffd10,$07a00000,$05e008bf,$f00ffd10
 dc.l $08800ff0,$01100a90,$00000110,$08800000,$01100480,$00000120
 dc.l $04ffffff,$ff200400,$00000020,$07ffffff,$ffe00000,$00000000
floppyend:
;graphic of disk drive not used by NTSC system
;Fast Copy

DecodeCnt:      dc.w 0      ;Counter for longwords to be decoded
DecodeAdr:      dc.l 0      ;Address where to decode
BlockAdr:       dc.w 0      ;Offset in Block for decoding
FTestAdr:       dc.l 0      ;Test address if Block already loaded
SectNum:        dc.w 0      ;Counter for Sector number
BytesBefGap:    dc.w 0      ;Bytes before the Gap
BytesAftGAp:    dc.w 0      ;Bytes after the Gap
FirstBlock:     dc.w 0      ;Block number of the first Block
FirstBlockSp:   dc.w 0      ;permanent memory for first Block
SectBL:         dc.w 0      ;Sector counter for Sectors before Gap
VerifyFlag:     dc.w 0      ;indicates if read or Verify
VerErrFlag:     dc.w 0      ;Flag for Errors
                            ;Verify-Bit = 1 => Error
TNumBufferA:    dc.w 0      ;Track-Number memory
                            ;for 1D-Copy (Start-Track)
TNumBufferE     dc.w 0      ;Buffer for End number
                            ;of loaded Tracks
BlockMessage:   ds.w 11

;Cruncher

Length:          dc.w $1600
ShrtByte:        dc.b 0
                align.w
TrackPointer:    ds.l 164   ;memory for Pointer to packed Tracks
MemoryBeg:       dc.l 0     ;memory beginning for crunching
MemoryLength:    dc.l 0     ;memory length for Crunching
MemoryChip:      dc.l 0
LengthChip:      dc.l 0
FreeFlagCh:      dc.w 0
MemoryFast:      dc.l 0
LenghtFast:      dc.l 0
FreeFlagFa:      dc.w 0
Subtr:           dc.w 0     ;See CrunEnd2
;Control

TrackBuffer1:   dc.l 0
TrackBuffer2:   dc.l 0
TrackNumS:      dc.w 0
TrackNumD:      dc.w 0
TrackNum:       dc.w 0
StartTrack:     dc.w 0      ;first Track to be read
EndTrack        dc.w 0      ;last Track to be read
MotorBits:      dc.b $F3
MotorBitsS:     dc.b $F3
MotorBitsD:     dc.b $F3
```

```
Flag:           dc.b $00    ;indicates if Source or Dest.
                align.w

;Deep Copy

Position:       dc.l 0      ;For Search routine
BitShifts:      dc.w 0      ;number of Bits during shift
Size1:          dc.w 0      ;size of the largest Block
Size2:          dc.w 0      ;size of the second largest Block
SizePos:        dc.l 0      ;Position of the largest Block
Searchln:       dc.w 0      ;how many defective Words can there be
NumWords:       dc.w 0      ;how many Words are compared
ErrorFlag:      dc.w 0
EndPos:         dc.l 0      ;End position of Track
BegPos:         dc.l 0      ;beginning of Track (after Gap)
SLength:        dc.w 0      ;number of Bytes to be written
WriteAddrs:     dc.l 0      ;Address from which writing starts
Offset:         dc.w 0     ;number bytes before Sync during writing
TrackBytes:     dc.w 0      ;Bytes on the Track
CheckLength:    dc.w 0      ;Bytes on the Track (Controll)
LenghtDest:     dc.w 0      ;Track length of Dest.-Disk
Syncwidth:      dc.w 0      ;Distance from Index to Sync
SyncNum:        dc.w 0      ;number of Syncs found
CopyTry1:       dc.w 0      ;Illegal Data, read how many times
CopyTry2:       dc.w 0      ;how many times read for NoSync
Blocks:         ds.l SortBlockNum ;memory for Blocklength+SyncNum
SortBlocks:     ds.l SortBlockNum ;memory for Block sorting
NumSortBlock:   dc.w 0      ;number of different sorted Blocks
SyncWord:       dc.w $4489  ;Value for reading
;Table for Sync-Search
;Sync = $4489
SyncBase:       dc.l %01000100100010010000000000000000
                dc.l %00100010010001001000000000000000
                dc.l %00010001001000100100000000000000
                dc.l %00001000100100010010000000000000
                dc.l %00000100010010001001000000000000
                dc.l %00000010001001000100100000000000
                dc.l %00000001000100100010010000000000
                dc.l %00000000100010010001001000000000
                dc.l %00000000010001001000100100000000
                dc.l %00000000001000100100010010000000
                dc.l %00000000000100010010001001000000
                dc.l %00000000000010001001000100100000
                dc.l %00000000000001000100100010010000
                dc.l %00000000000000100010010001001000
                dc.l %00000000000000010001001000100100
                dc.l %00000000000000001000100100010010
                dc.l %00000000000000000100010010001001

AmigaTrack:     dc.b 0  ;indicates if Amiga-Format on DeepCopy

fc:  dc.b 00    ;First Cylinder
lc:  dc.b 79    ;Last    "
tr:  dc.b 3     ;Tries
ws:  dc.b 0     ;Write repeatedly
vd:  dc.b 1     ;Verify
```

```
fa:  dc.b 1      ;Fast Copy
dc1: dc.b 0      ;D1 Copy
dc2: dc.b 0      ;D2 Copy
sd:  dc.b 1      ;SourceBits
dd:  dc.b 2      ;DestDits
sy:  dc.b 0      ;Sync correction

waiting: dc.b 0 ;irq wait_conter

DevName:        dc.b "trackdisk.device",0

                END
```

Index

Optional Diskette



For your convenience, the program listings contained in this book are available on an Amiga 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.

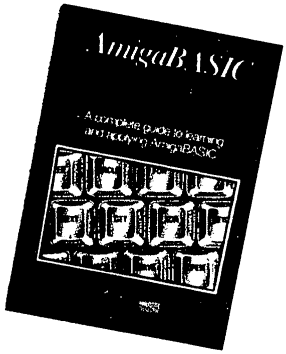
All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for $14.95 plus $2.00 ($5.00 foreign) for postage and handling.

When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

New Books for the AMIGA™

AMIGA BASIC- Inside and Out

AMIGA BASIC—Inside and Out is the definitive step-by-step guide to programming the AMIGA in BASIC. This huge volume should be within every AMIGA user's reach. Every AMIGA BASIC command is fully described and detailed. In addition, **AMIGA BASIC—Inside and Out** is loaded with real working programs.



Topics include:

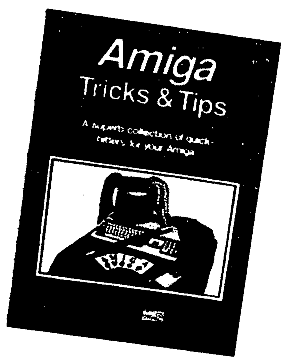
- Video titling for high quality object animation
- Bar and pie charts
- Windows
- Pull down menus
- Mouse commands
- Statistics
- Sequential and relative files
- Speech and sound synthesis

Plus much, much more. Over 550 pages of vital information are contained in this book. Available late first quarter 1987.

Suggested retail price: $24.95

AMIGA Tricks & Tips

AMIGA Tricks & Tips follows a tradition established by our other Tricks & Tips books for Commodore computers. It's another solid collection of diverse and useful programming techniques written for everyone who uses the AMIGA. This easy to understand source book details applications for the stunning processing power of the AMIGA.



Topics include:

- Displaying 64 colors on screen simultaneously
- Accessing libraries from BASIC
- Creating custom character sets
- Using Amiga DOS, graphics

In addition, **AMIGA Tricks & Tips** presents dozens of tips on windows, programming aids, the AMIGA's speech synthesis and musical capabilities, covers important 68000 memory locations, and much more.

AMIGA Tricks & Tips is available second quarter 1987.

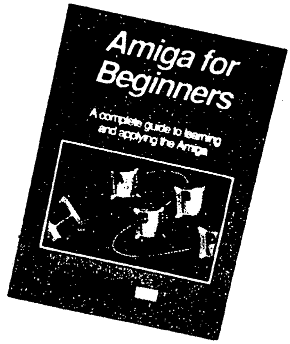
Suggested retail price: $19.95

Optional program diskettes are available for our *AMIGA* books
Suggested retail price: $14.95

New Books for the AMIGA™

Amiga for Beginners

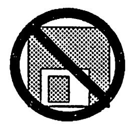
A perfect introductory book of you're a new or prospective Amiga owner. **Amiga for Beginners** introduces you to Intuition (the Amiga's graphic interface), the mouse, windows, the versatile CLI. This first volume in our Amiga series explains every practical aspect of the Amiga in plain English. Includes clear, step-by-step instructions for common Amiga tasks. Amiga for Beginners is all the info you need to get up and running.



Topics include:

- Unpacking and connecting the Amiga components
- Starting up your Amiga
- Customizing the Workbench
- Exploring the Extras disk
- Taking your first steps in the AmigaBASIC programming language
- AmigaDOS functions
- Using the CLI to perform "housekeeping" chores
- First Aid, Keyword, Technical appendixes
- Glossary

Suggested retail price: $16.95



No optional disk available

Amiga System Programmer's Guide

Amiga System Programmer's Guide has a wealth of information about what goes on inside the Amiga. Whether you want to know about the Amiga kernal or DOS commands, Amiga System Programmer's Guide has the information you need, explained in a manner that you can easily understand. Just a few of the things you will find inside:

- EXEC Structure
- Multitasking functions
- I/O management through devices and I/O request
- Interrupts and resource management
- RESET and its operation
- DOS libraries
- Disk management
- Detailed information about the CLI and its commands
- Much more—over 600 pages worth

Suggested retail price: $34.95

Optional program diskettes are available for our *AMIGA* books
Suggested retail price: $14.95

New Books for the AMIGA™

Amiga Machine Language

The practical guide for learning how to program your Amiga in ultrafast machine language. Used in conjunction with our AssemPro Amiga software package, **Amiga Machine Language** is a comprehensive introduction to 68000 assembler/machine language programming. Topics include:

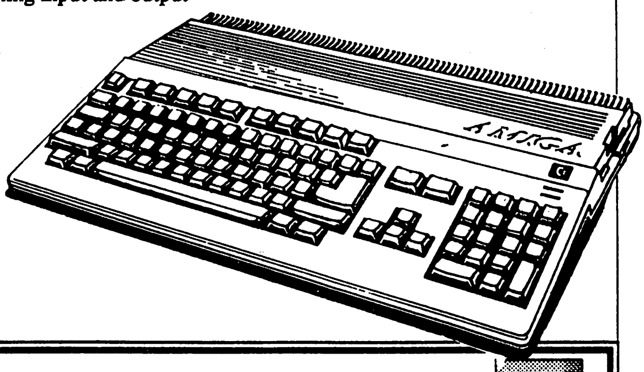
- 68000 microprocessor architecture
- 68000 address modes and instruction set
- Acessing RAM, operating system and multitasking capabilities
- Details the powerful Amiga libraries for using AmigaDOS
- Speech and sound facilities from machine language
- Many useful programs listed and explained

Suggested retail price: $19.95

AmigaDOS Inside & Out

AmigaDOS covers the insides of AmigaDOS from the internal design up to practical applications. There is also a detailed reference section which helps you find information in a flash, both alphabetically and in command groups.
Topics include:

- 68000 microprocessor architecture
- AmigaDOS - Tasks and handling
- Detailed explanations of CLI commands and their functions
- DOS editors ED and EDIT
- Operating notes about the CLI (wildcards, shortening input and output)
- Amiga devices and how the CLI uses them
- Batch files - what they are and how to write them
- Changing the startup sequence
- AmigaDOS and multitasking
- Writing your own CLI commands
- Reference to the CLI, ED- and EDIT commands
- Resetting priorities - the TaskPri command
- Protecting your Amiga from unauthorized use



Suggested retail price: $19.95

Optional program diskettes are available for our *AMIGA* books
Suggested retail price: $14.95

BeckerText

Powerful Word Processing Package for the Amiga

BeckerText ***Amiga*** is more than just a word processor.

BeckerText ***Amiga*** gives you all of the easy-to-use features found in our **TextPro** ***Amiga***, plus it lets you do a whole lot more. You can merge sophisticated IFF-graphics anywhere in your document. You can hyphenate, create indexes and generate a table of contents for your documents, automatically. And what you see on the **BeckerText** screen is what you get when you print the document—real WYSIWYG formatting on your Amiga.

But **BeckerText** gives you still more: it lets you perform calculations of numerical data within your documents, using flexible templates to add, subtract, multiply and divide up to five columns of numbers on a page. **BeckerText** can also display and print multiple columns of text, up to five columns per page, for professional-looking newsletters, presentations, reports, etc. Its expandable built-in spell checker eliminates those distracting typographical errors.

BeckerText works with most popular dot-matrix and letter-quality printers, and even the latest laser printers for typeset-quality output. Includes comprehensive tutorial and manual.

BeckerText gives you the power and flexibility that you need to produce the professional-quality documents that you demand.

When you need more from your word processor than just word processing, you need **BeckerText** ***Amiga***.

Discover the power of **BeckerText**.

Suggested retail price: $150.00

Features

- Select options from pulldown menus or handy shortcut keys
- Fast, true WYSIWYG formatting
- Bold, italic, underline, superscript and subscript characters
- Automatic wordwrap and page numbering
- Sophisticated tab and indent options, with centering and margin justification
- Move, Copy, Delete, Search and Replace
- Automatic hyphenation, with automatic table of contents and index generation
- Write up to 999 characters per line with horizontal scrolling feature
- Check spelling as you write or interactively proof document; add to dictionary
- Performs calculations within your documents—calculate in columns with flexible templates
- Customize 30 function keys to store often-used text and macro commands
- Merge IFF graphics into documents
- Includes ***BTSnap*** program for converting text blocks to IFF graphics
- C-source mode for quick and easy C language program editing
- Print up to 5 columns on a single page
- Adapts to virtually any dot-matrix, letter-quality or laser printer
- Comprehensive tutorial and manual
- Not copy protected

Selected Abacus Products for the *Amiga* computers

DataRetrieve

A Powerful Database Manager for the Amiga

Imagine a powerful database for your Amiga: one that's fast, has a huge data capacity, yet is easy to work with.

Now think **DataRetrieve** ***Amiga***. It works the same way as your Amiga—graphic and intuitive, with no obscure commands. You quickly set up your data files using convenient on-screen templates called masks. Select commands from the pulldown menus or time-saving shortcut keys. Customize the masks with different text fonts, styles, colors, sizes and graphics. If you have any questions, Help screens are available at the touch of a button. And **DataRetrieve's** 128-page manual is clear and comprehensive.

DataRetrieve is easy to use—but it also has professional features for your most demanding database applications. Password security for your data. Sophisticated indexing with variable precision. Full Search and Select functions. File sizes, data sets and data fields limited only by your memory and disk storage space. Customize up to 20 function keys to store macro commands and often-used text. For optimum access speed, **DataRetrieve** takes advantage of the Amiga's multi-tasking.

You can exchange data with **TextPro** ***Amiga***, **BeckerText** ***Amiga*** and other packages to easily produce form letters, mailing labels, index cards, bulletins, etc. **DataRetrieve** prints data reports to most dot-matrix & letter-quality printers.

DataRetrieve is the perfect database for your Amiga. Get this proven system today with the assurance of the Abacus **30-day MoneyBack Guarantee.**

Suggested retail price: $79.95

Features

- Select commands and options from the pulldown menus or shortcut keys
- Enter data into convenient screenmasks
- Enhance screen masks with different text styles, fonts, colors, graphics, etc.
- Work with 8 databases concurrently
- Define different field types: text, date, time, numeric & selection
- Customize 20 function keys to store macro commands and text
- Specify up to 80 index fields for *superfast* access to your data
- Perform simple or complex data searches
- Create subsets of a larger database for even faster operation
- Exchange data with other packages: form letters, mailing lists etc.
- Produce custom printer forms: index cards, labels, Rolodex•cards, etc. Adapts to most dot-matrix & letter-quality printers
- Protect your data with passwords
- Get Help from online screens
- Not copy protected

- Max. file size
- Max. data record size
- Max. data set
- Max. no. of data fields
- Max. field size

Limited only by your memory and disk space

Selected Abacus Products for the *Amiga* computers

TextPro

The Ideal Word Processing Package for the Amiga

TextPro *Amiga* is an full-function word processing package that shares the true spirit of the Amiga: easy to use, fast and powerful—with a surprising number of "extra" features.

You can write your first **TextPro** documents without even reading the manual. Select options from the dropdown menus with your mouse, or use the time-saving shortcut keys to edit, format and print your documents.

Yet **TextPro** is much more than a beginner's package. It has the professional features you need for all of your printed documents. Fast formatting on the screen: bold, italic, underline, etc. Centering and margin justification. Page headers and footers. Automatic hyphenation of text. You can customize the **TextPro** keyboard and function keys to suit your own style. Even merge IFF-format graphics right into your documents. **TextPro** includes ***BTSnap***, a utility for saving IFF graphics that you can use in your graphics programs. This package can also convert and print other popular word processor files.

TextPro is output-oriented. This means you can print your documents to exact specifications—and get top performance out of your dot-matrix or letter quality printer. (Printer drivers included on diskette let you customize **TextPro** to virtually any printer on the market). The complete tutorial and manual shows you how it's all done, step by step.

TextPro sets a new standard for word processors in its price range. Easy to use, packed with advanced features—it's the Ideal package for all of your wordprocessing needs. Backed by the Abacus **30-day MoneyBack Guarantee.**

Suggested retail price: $79.95

Features

- Fast editing and formatting on screen
- Display bold, italic, underline, superscript and subscript characters
- Select options from dropdown menus **or** handy shortcut keys
- Automatic wordwrap & page numbering
- Sophisticated tab and indent options, with centering & margin justification
- Move, Copy, Delete, Search &Replace options
- Automatic hyphenation
- Customize up to 30 function keys to store often-used text, macro commands
- Merge IFF format graphics into your documents
- Includes ***BTSnap*** program for saving IFF graphics from any program
- Load & save files through RS-232 port
- Flexible, *ultrafast* printer output—printer drivers for most popular dot-matrix & letter quality printers included
- Comprehensive tutorial and manual
- Not copy protected

How to Order

Abacus 5370 52nd Street SE Grand Rapids, MI 49508

All of our Amiga products–application and language software, and our Amiga Reference Library–are available at more than 2000 dealers in the U.S. and Canada. To find out the location of the Abacus dealer nearest you, call:

Toll Free 1-800-451-4319

8:30 am-8:00 pm Eastern Standard Time

Or order from Abacus directly by phone with your credit card. We accept Mastercard, Visa and American Express.

Every one of our software packages is backed by the **Abacus 30-Day Guarantee**—if for any reason you're not satisfied by the software purchased directly from us, simply return the product for a full refund of the purchase price.

Order Blank

Send your completed order blank to:

Abacus Software, Inc.
5370 52nd St. S.E.
Grand Rapids, MI 49508

Your order will be shipped within 24 hours of our receiving it

Name: ____________________

Address: ____________________

City ________ State ________ Zip ________ Country ________

Phone: ____ / ____________

Qty	Name of product	Price
	Mich. residents add 4% sales tax	
	Shipping/Handling charge (Foreign Orders $12 per item)	
	Check/Money order TOTAL enclosed	

Credit Card#

Expiration date

For extra-fast 24-hour shipment service, order by phone with your credit card

VISA · MasterCard · American Express Card

We appreciate your selection of another of our fine products.

In addition you may receive the Abacus on Amiga Newsletter

FREE

Return this completed card to receive Abacus on Amiga, our newsletter that keeps you informed about Abacus' newest products for the AMIGA.

Reserve your copy now!

Abacus on Amiga

Abacus
5370 52nd Street S.E., Grand Rapids, MI 49508
(616) 698-0330

Name ____________________________________

Address ____________________________________

City ________________ State ________ Zip __________

Where did you purchase AMIGA DISK DRIVES: Inside and Out ? ________________

What other Abacus Products would you be interested in? ________________

Amiga Disk Drives Inside & Out

Amiga Disk Drives: Inside & Out shows everything you need to know about Amiga disk drives. You'll find information about data security, disk drive speedup routines, disk copy protection, boot blocks and technical aspects of the hardware.

If you're a beginner, you'll learn simple disk drive operations using AmigaBASIC, the Workbench and the CLI (Command Line Interpreter). You'll also learn about loading and saving programs; sequential and relative file management techniques and much more.

If you're an advanced user, you'll see how to access the disks without DOS, how to use and change the full-length disk monitor so that you can explore the disk by track and sector. This way you'll discover the inner workings of the disk drive.

Amiga Disk Drives: Inside & Out topics include: disk drive operations • file management • CLI, Workbench, AmigaBASIC disk commands and functions • disk and file copy methods and techniques • protecting files • disk format information

FastCopy: You can copy entire diskettes in one minute!
DeepCopy: Backup many copy protected disks Also copy ST and PC 3 1/2" disk formats with your Amiga disk drive!
FloppySpeeder: Super fast-track disk routines independent of diskette access
CrunchCopy: Packing routines without changing disk formats
Disk Monitor: Full-length track and sector editor

The most thorough coverage of Amiga disk drives ever.

Many real, working programs presented described and explained:

- AmigaBASIC: loading, saving data and sequential and relative file management
- Floppy disk operations from the Workbench and CLI (Command Line Interpreter)
- DOS functions and operations
- Disk block types, boot blocks, checksums, file headers, hashmarks and protection
- Viruses and how to protect your boot block
- Trackdisk.device: Commands and structures
- Trackdisk-task: Function and design
- Diskette access with DOS (Disk Operating System)
- MFM, GCR, track design, blockheader, datablocks, checksums, coding and decoding data, hardware registers, SYNC and interrupts

Optional Program Diskette available:
Contains every program listed in the book - complete, error-free and ready to run! Saves you hours of typing in program listings.

ISBN 1-55755-042-5

Abacus 5370 52nd Street SE • Grand Rapids, MI 49508