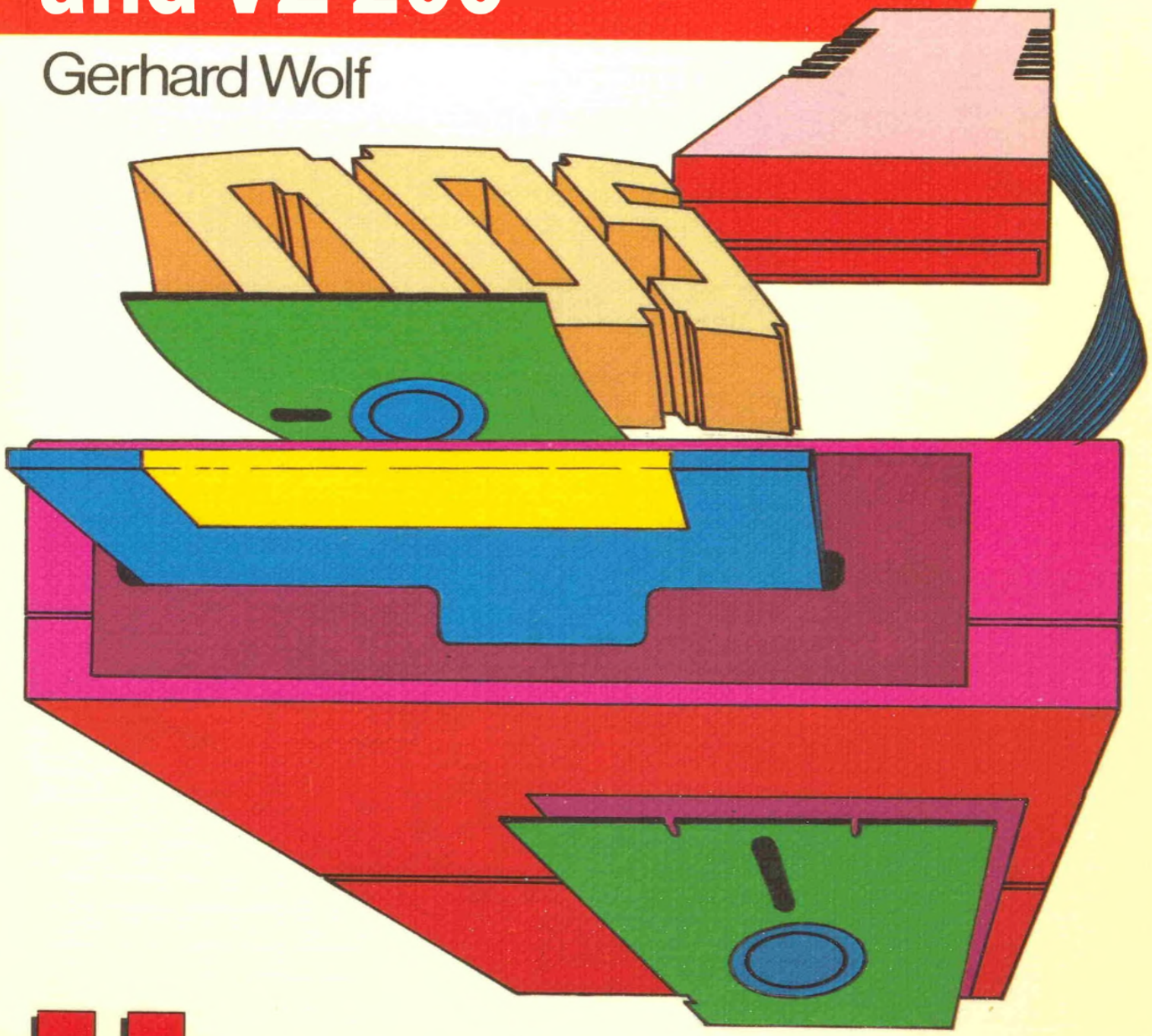
active and creative computing

The Laser DOS for Laser 110, 210, 310 and VZ 200

Gerhard Wolf



My Home-Computer

Gerhard Wolf

The Laser DOS for Laser 110, 210, 310 and VZ 200

HC - My Home computer

Gerhard Wolf

The Laser DOS for Laser 110, 210, 310 and VZ 200

Structure and application of Floppy Operating System

VOGEL BOOK PUBLISHING
WURZBURG

Published by the same author:
ROM listings for Laser 110, 210, 310 and VZ 200

(HC - My Home Computer)
ISBN 3-8023-0852-2

The BASIC interpreter in the Laser 110, 210, 310 and VZ 200

(HC - My Home Computer)
ISBN 3-8023-0874-3

CIP short title recording of the German Library

Wolt, Gerhard:
The Laser DOS for Laser 110, 210, 310 and VZ 200: structure and application floppy disk operating systems / Gerhard Wolf. - 1st edition - Würzburg: Vogel, 1985.
(HC - My Home Computer)
ISBN 3-8023-0868-9

ISBN 3-8023-0868-9
1st edition. 1985

Printed in Germany

Cover design: Bernd Schröder, Böhl
Manufacture: Alois Erdl KG, Trostberg


LASER
COLOR COMPUTER 210
CTRL
SHIFT
RETURN
SPACE


LASER
COLOR COMPUTER 310
RETURN
CTRL
SHIFT

0. Introduction

In order to understand and see through something, solid basic knowledge is essential. Only then will you be able to recognize connections and use them correctly.

This principle also applies - or even more so - to the use of a floppy disk systems on a home computer, which one can assume is not the first line was procured for commercial purposes, but primarily to accommodate a hobby to indulge in or to get further training in the field of data processing.

This book deals specifically with the connection of a floppy disk system to a LASER 110, 210, 310 or even VZ200.

It describes in detail the basics and structure of the system and deals with detailed with the possibilities of the supplied disk drive system {Disk Operating System = DOS).

It is intended for all floppy disk users, whether they are new to computers or more experienced "Freak",

In the first chapters, the basics are presented and the use within the available BASIC language is described. Then they follow byte (partially bit) precise description of the data organization on the diskette and a detailed explanation of the diskette possibilities for assembler and machine program - experts.

However, don't let this put you off if this is your first time using a diskette. The structure of this book allows step-by-step climbing into matter.

After studying the basics, start using the common DOS instructions and the program/file management by first only using the floppy disk and use it as a storage medium for your programs.

If you are sure about this, try using your own data from BASIC programs on the disk and then process it again.

You should only turn to the last two chapters if the DOS application in BASIC is completely familiar to you and you have some basic knowledge of the Z80 Assembler and machine language programming skills.

1. The LASER 110, 210 and 310 Disk System

Components

The floppy disk drive **LASER DD20** was specially designed for one of developed LASER computers 110, 210 and 310; However, it is suitable also for the VZ200. These floppy disk drives are not compatible with others computer systems (e.g. the LASER 2001 or LASER 30000 etc). Conversely, other floppy disk drives are also not suitable for these computers.

The connection to the computers is established with the help of a floppy disk controller (Floppy Disk Controller) **LASER DI40**, which is connected to the System Bus off of the computers and a memory expansion can be plugged into the "piggyback" (Figure 1.1).

Two drives can be connected to a floppy disk controller at the same time to operate. The floppy disk controler also contains the necessary system programs extension (Disk Operating System = DOS)

On the LASER 110 computer and on the VZ200 with internal 4K-RAM there is required an additional memory expansion of at least 16K.

Figure 1.1 LASER 310, floppy disk controller, 16K memory expansion and drive

Basics of floppy disk storage

The drive

There are a large number of different floppy disk systems of all kinds manufacturers. One of the main classifications of these drives is the size of the floppy disks. These vary from 3 1/2 inches to 5 1/4 inches to 8 inches. The floppy disk drive **LASER DD20** is a 5 1/4 inch Drive, which is very sdimilar in terms of presentation and technical structure to TEAC drives.

Floppy disk drives work with a round, rotating disc (called floppy disk) which, like a tape, is covered with a magnetizable layer. The data is written to these or read from again using a read/write head.

In order to be able to access any position on this disk, the read/write head has to be movable. For this purpose it is mounted on a rail and can be moved across the disc.

To write or read data, the head is simply moved in or out the desired distance and then waits for the desired data to rotate past underneath it (Figure 1.2).

For the sake of clarity and retrieval, the head has fixed grid positions, which in turn create concentric circles of data on the disk. The various disk systems have between 35 and 88 grids. In **LASER DD20** there are 40.

The resulting circles of data on the floppy disk are called tracks. Each individual track can be accessed directly and precisely by the adjustment mechanism of the head.

Within a track, the data is recorded bit by bit one after the other, that is "sequentially" in computer terms. As a result, after the head has been positioned over the track, it naturally takes an average of half a turn of the disc before the desired data is reached .

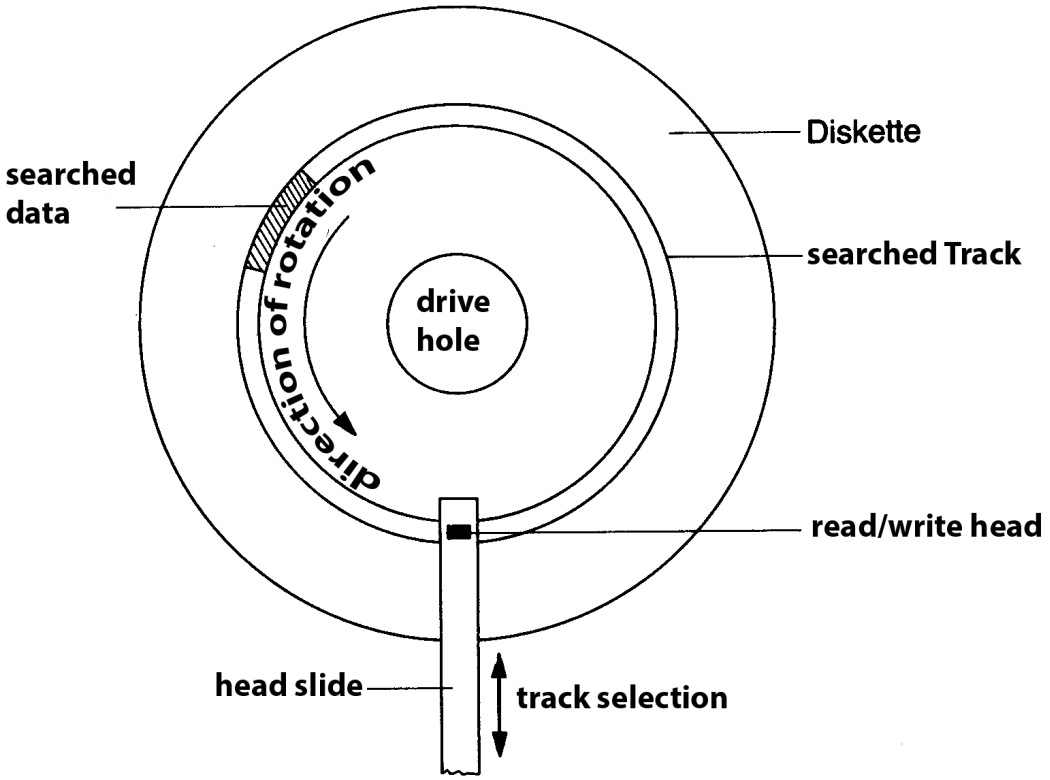


Figure 1.2 Data access on a floppy disk

The time it takes to access the data you want depends on how fast the head can be positioned on the particular track and how fast the disk is spinning.

With the **LASER DD20**, the diskette is driven at 80 revolutions per minute. The time to move the head from one track to the next is about 20 milliseconds This results in an average access time of 500 milliseconds.

During a read/write operation, the floppy disk is held firmly like an audio tape pressed against the read/write head. This is covered with a piece of felt works, which presses the diskette from above onto the head via a lever macaw. This lever arm is connected to the locking lever on the front of the drive. If this is in a vertical position (closed), the diskette is upside down pressed; in the horizontal position, the diskette is free (Figure 1.3).

An electronic head-loading procedure is common to many other drives which does not exist here.

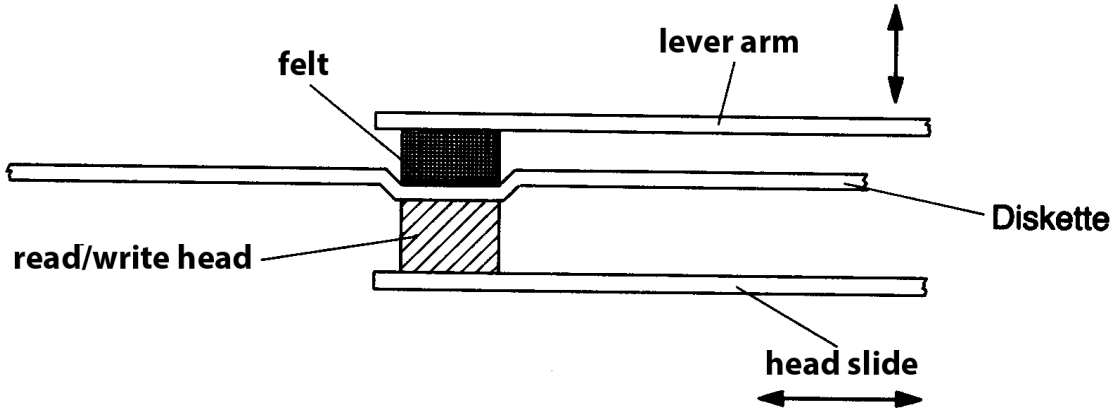


Figure 1.3 Representation of a floppy disk in the drive

If you've been paying attention, you must have noticed that a floppy disk is always written from the bottom up, which is different from how you actually think it is when you put it in.

The drive mechanism is also connected to the locking lever. Closing the lever (vertical) centers the disk drive hole on a cone driven by the motor via a belt. How exactly a disk is centered on the cone is one of the critical components of the drive.

The position of a track always refers to the center of the diskette. Therefore, reliable writing and retrieval of the data depends very much on how precisely the diskette is centered. Unfortunately, the **LASER DD20** drive tends to not press the floppy disks precisely onto the canoes. Please note the help and control options mentioned in the "Inserting a floppy disk" section,

To protect the floppy disks, the drive motor is only switched on immediately before the write or Read operations - turned on and then immediately turned off again. You will receive a visual indication of this process in the form of a lit LED (light emitting diode) on the front of the drive. If this lights up, you should not remove or insert a disk (see head pressing process).

The floppy disk

Construction

Due to the fact that the disk is pressed against the read/write head, it is evident that no solid disk can be used as the base material. A flexible, dimensionally stable plastic material (usually a mylar sheet) is used that is coated with iron oxide. This is where the English name "floppy disk" comes from.

For better handling and to protect the coating, the disc is packed in a rigid sleeve with a felt-like lining.
The lining fulfills three essential tasks:

- Reducing friction between disk and case
- Dissipate static charges caused by rotation
- Absorption of dust and dirt particles that have penetrated to protect the coating

The diskette stays in this sleeve at all times, which has three openings for handling:

- a large hole in the middle for the drive mechanism
- an oval cutout through which the head on the coating can access
- a small opening as an index hole, which is used by many drives to physically identify the start of a track (not by the **LASER DD20**).

There is a small rectangular recess on the back of the case. This is device write-protection that you can use to protect the contents of a floppy disk. If this gap is covered with one of the adhesive strips supplied with the floppy disk packaging, writing to this floppy disk is blocked. (Figure 1.4).

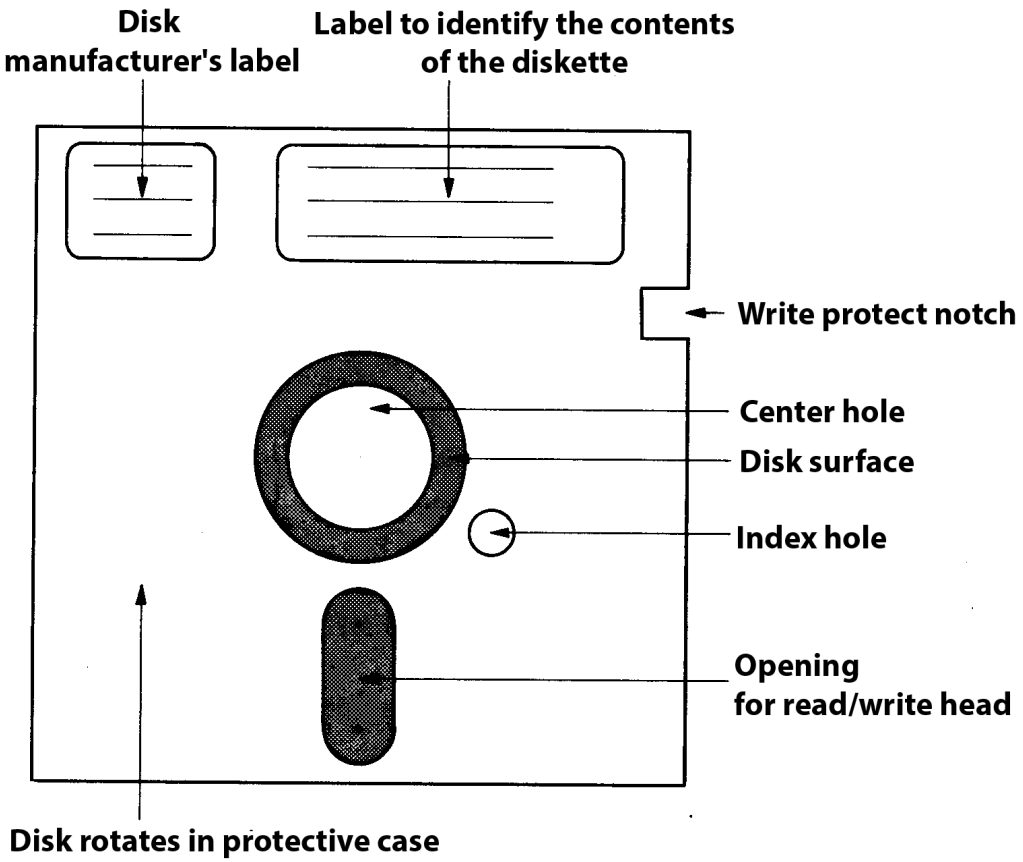


Figure 1.4 Floppy disk

The actual data carrier only consists of a very thin coating of iron oxides and is specially treated in order to be able to remain in contact with the read/write head for a reasonable length of time. Nevertheless, this layer is abraded over time with constant head contact. A running time of 50 - 60 hours can serve as a guide. This results in a fairly long service life, since the disk only runs for read/write access and the head is not always on the same track.

Because of this abrasion, malicious gossip also refers to diskette processing as "cutting data technology".

Insert the disk

To insert a floppy disk, the locking lever on the drive opening is set horizontally. The floppy disk is now pushed into the drive until it stops, with the oval head cutout to the front and normally with the label facing up.

Turning the locking handle to the vertical position presses the drive hole on the floppy disk centered on the drive cone of the drive and presses the floppy disk against the read/write head.

Normally this is sufficient and the read/write operations can begin.

However, as previously mentioned, difficulties can arise when the disk is crookedly clipped onto the cone, and unfortunately this often happens with the **LASER DD20**.

You should therefore make the following points a rule:

- Use only floppy disks with a reinforcement ring around the drive shaft. This prevents premature wear on the drive hole due to crooked pinching on the cone.

- Before inserting the disk, center it by hand as best you can in the case.

- Before writing anything to an initialized floppy disk, test it with the DIR command.

- After reinitialization, remove the diskette, reinsert it and also test with the DIR command.

If, despite this, the Drive keeps trying to reposition the head during a read process (you can hear this from the slightly louder clicking noises), open the locking lever briefly with the motor running and close it again immediately. As a rule, the disk pulls cleanly onto the cone when the engine is running.

A floppy disk has two sides

Which diskettes do you now obtain for the **LASER DD20**?

As mentioned, this is a 5 1/4 inch drive, so you will also need 5 1/4 inch floppy disks. But there are also many different types, hard sectored or soft sectored, one-sided or double-sided, with single density or with double density,

To sum it up! You will need 5 1/4 inch, soft sectored, single sided, single density floppy disks. These bear the designation "SSSD"; this stands for "single sided, single density".

You can also use double-density floppy disks (SSDD). These have been tested a little more thoroughly, are a bit more expensive, but are not absolutely necessary for the recording process used with the **LASER DD20**.

If you look closely at such a "single-sided" diskette, you will notice that both sides are coated and that the oval opening for the head is also present on both sides of the case.
This means that in principle you can write to both sides of the disk.

However, the **LASER DD20** drive is equipped with only one read/write head. So you had to turn the disk to bring it to the other side. If you do this and now try to write something on this side, you will get the message "?DISK WRITE PROTECTED". This is due to the lack of a write protect notch on the other side of the case. Remember that a floppy disk is write-protected when you tape over the write-protect notch.
“No write-protect notch” obviously has the same effect.

In order to be able to use the second side, only a second write protection notch is required, which you can easily attach to the case with a hole punch. Use another disk as a template. Don't worry about damaging the disk itself, it doesn't reach that far into the corners of the case (Figure 1.5).

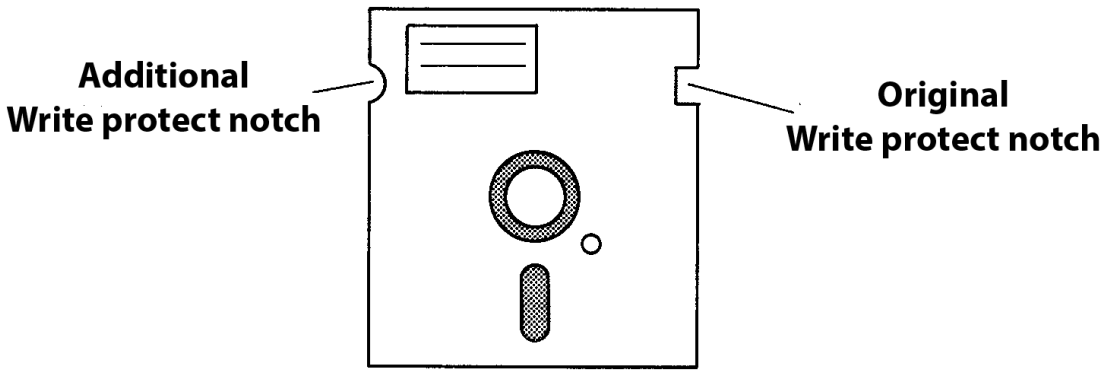


Figure 1.5 Double-sided use of the floppy disk

You have now created 80,000 bytes of additional storage space per diskette. However, you should turn the disk over if you want to read or write to the back.

Handling of floppy disks

In order to protect your data on the floppy disks from destruction as much as possible, you should absolutely observe the following rules:

- Always keep floppy disks in their protective cases when they are outside the drive.

- Make sure there is no disk in the drive when turning the power on or off.

- Never bring your floppy disks close to strong magnetic fields (transformers, motors, magnets, televisions/monitors, radios, etc.); the magnetic fields emitted there could destroy the data content.

- Only touch the disk by the sleeve. Avoid touching the magnetizable coating. Also try to clean the coating. Scratches are quick on the surface and you can then forget about the disk.

- Never expose a floppy disk to direct sunlight or excessive heat.

- Avoid contaminating the coating with cigarette ash, dust or other things.

- Only use a fiber pen if you want to write on the label on the case. Ballpoint pens or pencils could damage the coating through the case.

- Whenever possible, store floppy disks upright (like vinyl records) so that there is no pressure on the sides.

Tips on disk labeling.

Each floppy disk has a label permanently affixed to its case. You should only use this for important information that does not change during the life of a floppy disk. For example, it is very helpful to give the diskettes a consecutive number for archiving. This would have its best place there. Other useful data include your name and the date the disk was first used.
-

For information on contents, it is best to use the adhesive labels enclosed with each pack of disks, which you can also easily change once in a while. If you do not use them to seal any important openings, you can use the entire surface of the case for this purpose.

Record structure

What determines the amount of data that can be stored on a floppy disk? Each system has its own disk storage capacity; with the 5 1/4 inch floppy disks this is possible up to 1/2 million bytes (characters) per disk side. With the **LASER DD20** it is slightly more than 80000 bytes.

Two key factors affect storage capacity. This is the number of increments with which the head moves over the floppy disk and which is equal to the number of data tracks to be written on the floppy disk. There are currently Known to vary between 35 and 89 on different systems.
The **LASER DD20** has 40 tracks.
The second factor is the way each bit is written to disk. A distinction is made here between “single density” (FM) and “double density” (MFM). Double recording density also results in approximately twice the capacity. As already mentioned, the **LASER DD20** records with single density.

However, the storage capacity could be almost twice as large if the data were written to the floppy disk exactly as they are in the memory without any further measures being taken. With this you get a lot of data on the diskette, but you can't do much with it anymore. How could you find out a specific piece of information in a jumble of bits without having to go through everything from the beginning.

The benefits of disk storage are realized only when the records are organized in a meaningful way by breaking them into small manageable chunks that have a known location on the disk. This is the only way to take advantage of direct access. This means nothing other than that you have to format the recordings.

Such formatting is achieved by dividing the recording on the disk within the 40 different tracks into 16 equal sections (sectors), like a pie. Each of these sectors is separately addressable and can be treated individually.

Each track consists of 16 sectors, in each of which 128 data bytes can be accommodated (Figure 1.6). For the **LASER DD20** this means a precise storage capacity of

48 tracks x 16 sectors x 128 bytes = 81920 bytes

per disc side.

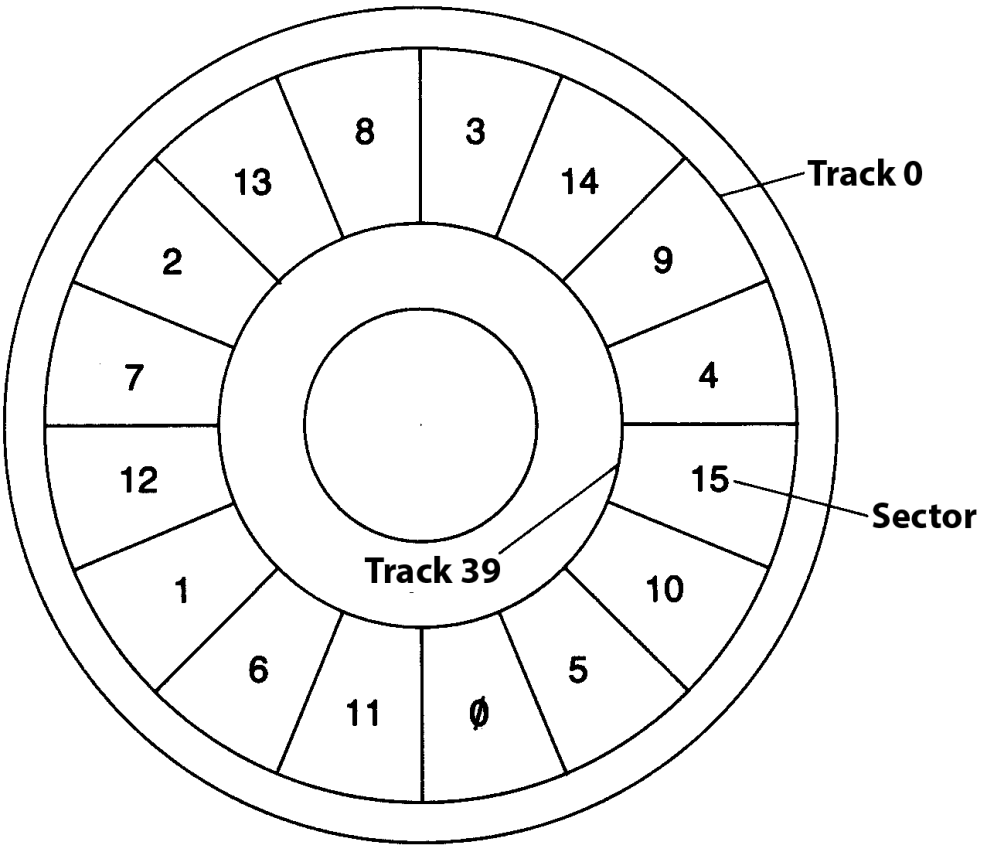


Figure 1.6 Arrangement of tracks and sectors on a floppy disk

However, this is not all that needs to be stored on a formatted floppy disk

Among other things, to be able to access a sector directly without any major fuss, man must know when the information you are looking for is passing under your head.

For this purpose, each sector receives a header, a so-called address field, in which the sector number and, in order to recognize head alignment errors, also the track number are noted.

To detect recording errors within a sector, a checksum field is added at the end of the sector.

But this alone is not enough. The head is rarely at the point where a new byte begins on the track. As a rule, it will start reading in the middle of a byte. However, since the data is stored consecutively bit by bit without gaps, it is impossible to identify the beginning of a byte. That is, first of all, a start of recording is found. One speaks here of a synchronization of the head.

For this purpose, specially defined bit sequences and recording marks are written onto the diskette, which have an easily recognizable pattern.

There are two different types of these marks. One precedes each sector address field, which is the "address mark"; a second precedes each data field of a sector, the "data mark".

Each of these markers are preceded by sync bytes, and the markers are immediately followed by the data. This allows one to clearly distinguish whether one is in front of a data record or in front of an address field.

Further space is lost on the floppy disk due to "recording gaps" located behind each data field of a sector. These gaps are urgently needed in order to be able to compensate for fluctuations in the rotational speed within certain limits (Figure 1.7).

Such a basic structure of the diskette must first be created before any data is written to it. This process is called "initialization"; a separate command is available for this. During initialization, the subdivision into sectors is carried out and all address and data marks are written.

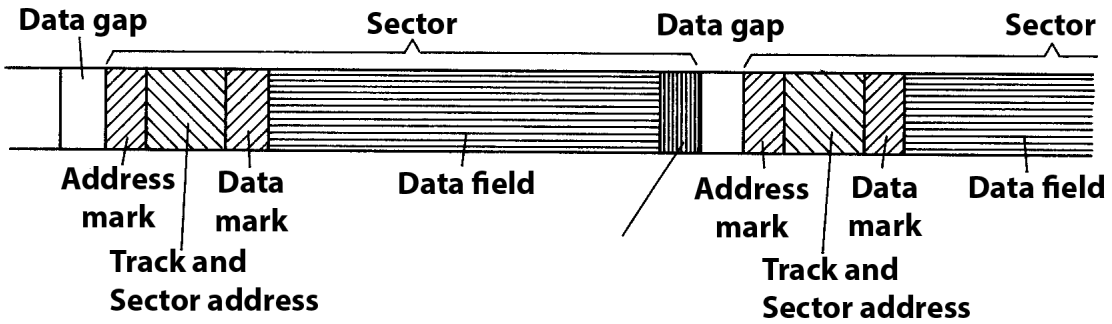


Figure 1.7 Data structure on a floppy disk

Figure 1.6 shows that the sectors are not numbered consecutively from 1 to 16, but in jumps on the diskette. With this little trick, it is possible to read several sectors in a row during one revolution of the diskette and thus speed up access considerably.

After these explanations it should be understandable how the computer can find every single sector on the diskette.
However, you usually don't want to know anything about individual sectors, you are looking for a specific program on the diskette or a file that you have created there. As a rule, you will also have more than one program or file stored on a diskette. How do you get such a complete record without having to keep track of sectors yourself?

A whole track of the floppy disk was sacrificed for this purpose. On track 0, the outermost track, there is a table of contents on the diskette, in which it is recorded which programs and files are stored on the diskette and where they can be found. With the DOS command "**DIR**" you can display this table of contents on the screen.

The last sector of this track 0 still has a special use. It notes whether each sector of the diskette is free or contains valid data.

The Floppy Disk Controller

A **LASER DI40** Floppy Disk Controller is required to connect a drive to a computer.

This is connected to the computer's system bus and has two 20-pin plug-in connections on the back for connecting one or two drives.

The task of the floppy disk control is the implementation of logical orders, such as "Read a program" in single steps for specific control of the drives, e.g. step pulses for track adjustment, read a bit, write a bit, motor on, motor off, etc.

The floppy disk control also contains the floppy disk operating system (Disk Operating System), DOS for short. This is stored on 8K ROMs and expands the BASIC language range of the computer with 17 commands that are required to operate the floppy disk station. These include **"INIT"** for diskette initialization, **"SAVE"** and **"LOAD"** for saving or loading a BASIC program.

This diskette operating system is addressed using addresses 4000 - 5FFF (hexadecimal), which have been reserved for this expansion purpose. A 310-byte work area is also reserved at the end of the RAM area.

Installation of the floppy disk system

In principle, all connections of electronic components should only be made or released when the power supply is switched off. Otherwise, how quickly is an expensive fully integrated component destroyed by voltage peaks that occur.

This also applies to the connection of the floppy disk system. So make sure that the power supply of the computer system is turned off.

The floppy disk controller is first connected to the system bus on the back of the computer. This is the plug strip to which a possibly existing memory extension was connected. You can now connect it to the top of the floppy disk controller.

There are two 20-pin connector strips on the back of the floppy disk controller
to connect the drives, they are marked "D1" for drive 1 and "D2" for drive 2

If you only have one drive, connect it to "D1".

Each drive requires a separate power supply. The 5-pin DIN plug of the power supply unit must be plugged into the corresponding socket on the back of the drive.

Now connect the power packs to a mains socket and your setup is complete.
You should make sure that there is no diskette in the drive when making or breaking the power connection, as this could possibly destroy the data content.

You should always disconnect the power supply from your floppy disk drives if you will not be using the system for a long period of time.

It is advisable to procure a 220 volt plug strip with an illuminated on/off switch for all mains connections of the computer system. This way you can interrupt the entire power supply when you have finished your work by simply pressing a button.

Remember that the LASER 110 and VZ200 computers with 4K internal RAM must have at least a 16K memory expansion.

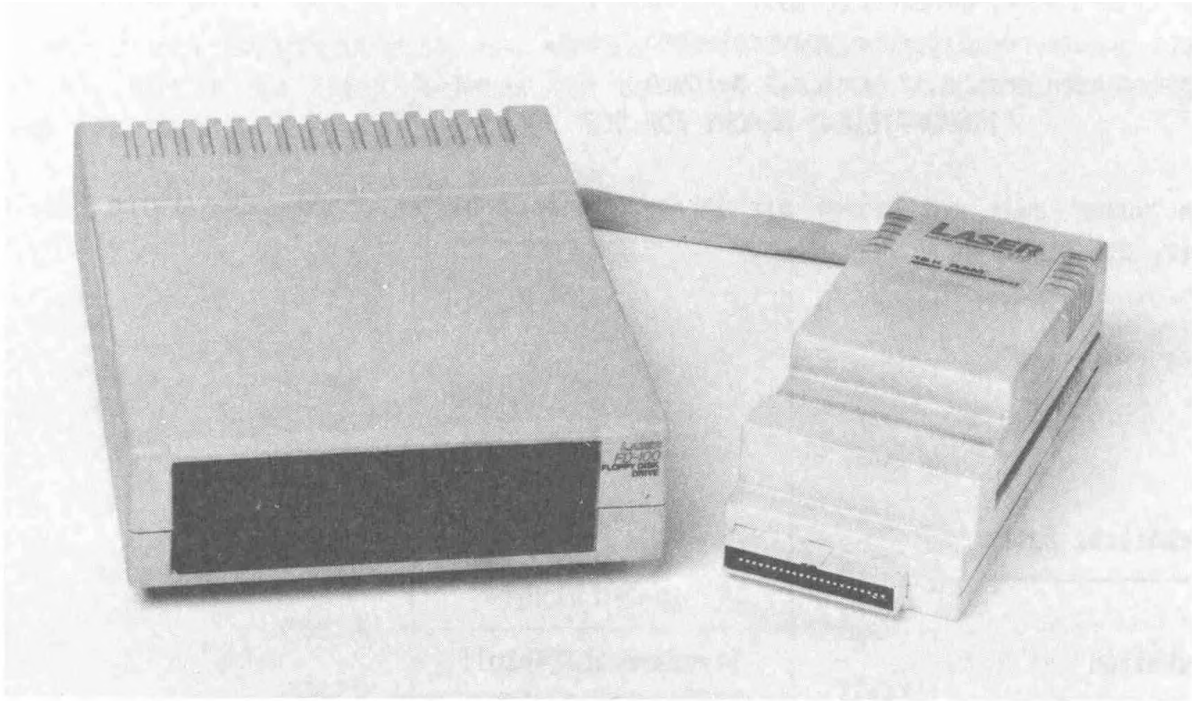


Figure 1.8 Floppy disk drive, 16K memory expansion and floppy disk control

System initialization

After successful installation, turn on your computer as usual.

The computer's initialization routine automatically recognizes that a floppy disk system is connected and briefly starts the motor of the drive connected to "D1". The read/write head is positioned on track 0 (that's the clicking noise).

The text **"BASIC V2.0"** or **"BASIC V1.2"**

is replaced by the text **“DOS BASIC V1.0”**

This is your sure sign that your diskette drive system is initialized.

You now have the additional 17 diskette editing commands mentioned above.

Of course, this does not happen if you have not connected any additional memory expansion via a LASER 110 or VZ200.
Instead, the message appears there

?INSUFFICIENT MEMORY FOR DOS”

You can work with your computer as before, but you have no way of accessing the floppy disk.

Technical specifications

Floppy Disks	-	Standard 5 1/4 inch SSSD (single sided, single density)
Recording format	-	single-sided single density 40 tracks 16 sectors/track 128 bytes/sector
Capacity	-	80 KB
Drive	-	80 revolutions / minute
Power supply	-	+5V, +12V DC voltage via separate power supply

2. DOS - operations

Structure of the LASER-DOS

The LASER floppy disk operating system, DOS for short (Disk Operating System), is an additional program package of approx. 8 KB that has been accommodated in ROM modules in the housing of the Floppy Ddisk Controller.

It includes the memory area from address 16384 (4000H) to address 24575 (5FFFH), which is kept free in the LASER computers for such expansion purposes (Figure 2.1).

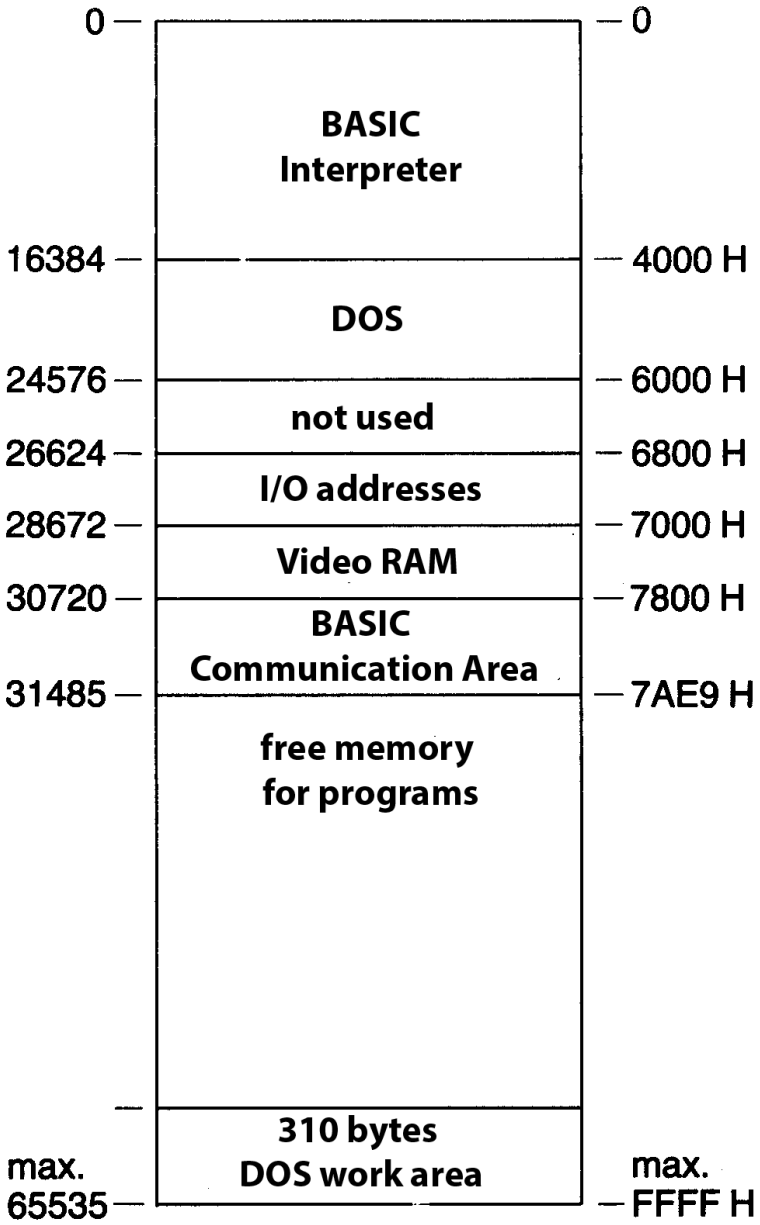


Figure 2.1 The memory allocation of the LASER 110, 210, 310 and the VZ200

When the computer is switched on, the existence of the floppy disk operating system is automatically recognized by the initialization routine of the standard ROM and initialized, i.e. embedded in the BASIC interpreter's sequence routines.

The floppy disk operating system has its own command interpreter, which independently recognizes the 17 additional input commands and initiates its own execution routines.

The additional commands are exclusively floppy disk operations that enable you to save and retrieve programs and also to manage your own data stocks on the floppy disk.

Use of DOS commands

Commands - Syntax

The available commands are entered without any special identification like usual BASIC statements.

Most of these commands can be used both in BASIC programs and in direct mode, i.e. for immediate execution from the screen.

However, a select few are limited to one mode or another. When exactly which command may be used is noted in the detailed descriptions.

In terms of syntax, the commands can be divided into three categories:

- Commands that do not address a file

  command [parameter]

- Commands that address a file

  command "filename" [,parameter]

- Commands that address two files

  command "filename1","filename2"

Parameters are additional information required by some commands. If several parameters are required, they must be separated by commas.

A small restriction arises when using it within BASIC programs.

The additional diskette commands are not recognized if they are specified directly after a **THEN** or **ELSE** in **IF** statements.

They must always be entered as an independent command either at the beginning of a line or after a command separator "**:**".

100 IF A= 1 THEN RUN "XYZ" wrong

100 IF A= 1 THEN :RUN "XYZ" correct

or

100 IF A <> 1 THEN 120
110 RUN "XYZ"
120......

File Types and Specifications

There are three different types of files in LASER-DOS:

- BASIC program files
  with the label "T" as file type (= text file).

  BASIC programs are stored on the diskette in this file type.

- Machine program files
  with the label "B" as file type (=binary file).

  Machine programs are stored on the diskette in this file type.

- Data files
  with the label "D" as file type (=data).

  Your personal data is saved in this file type if you want to store it on the diskette from a BASIC program.

BASIC and machine programs are stored on the diskette in the same format. The different type designation is only in the table of contents and causes different handling when loading and starting.

Data files have a completely different structure, which means that there are also restrictions when using individual commands.

If you want to address a file on the floppy disk or create a new one, you must specify a file name in the commands, which is entered in the table of contents of the floppy disk.

A file name can be a maximum of eight characters long and can consist of any sequence of letters, characters or numbers.

In the commands, the file name must always be given in quotation marks. In contrast to other BASIC commands, the final quotation mark must not be forgotten, even if no further information is given.

Unfortunately, LASER-DOS does not allow using a string variable instead of the file name; this must always be specified in full directly in the command. This complicates the flexible handling of different data files. How you can still help yourself is noted in chapter 5 "Tips for programming".

Commands - Overview

The 17 additional disk commands can be functionally divided into three groups.

General Instructions

INIT	Initialize a floppy disk. This command puts the basic structure on the diskette, i.e. it is divided into tracks and sectors.
DRIVE n	Drive selection. This allows you to select one of the two attachable drives for further processing.
DCOPY	Copy disks. With this command you copy the contents of one floppy disk to another.
STATUS	Output diskette status. With "STATUS" you can display the space still available on the diskette. (only from DISK BASIC V 1.2)

File Management Features

DIR	Output of the table of contents. All programs and files stored on the disk are listed on the screen.
SAVE "name"	Save a BASIC program. A BASIC program in memory is written to disk with the filename "name".
LOAD "name"	Load a BASIC program. The BASIC program marked with "name" is read from the diskette.

RUN "name" Load and start a BASIC program.
The BASIC program marked with "name" is read from the diskette and started immediately.

BSAVE "name",aaaa,eeee Saving a machine program
A machine program in the memory is written to the diskette with the file name "name".

BLOAD "name" Loading a machine program.
The machine program specified with "name" is read in from the diskette.

BRUN "name" Loading and starting a machine program.
The machine program specified with "name" is read in from the diskette and started.

REN "name1",”name2” Rename a file.
The file named "name1" will be renamed to "name2" on the disk.

ERA "name" Delete a file.
The file labeled "name" is deleted from the floppy disk.

DCOPY "name" Copy a program.
The BASIC or machine program identified by "name" is copied to another diskette.

Storage and processing of data

OPEN "name",n Open a data file.
The data file designated with "name" is opened for writing or reading.

PR# "name",var1[,var2…,varn] Write in a data file.
The variables specified in the command are combined into a data record and written to the data file designated with "name".

IN# "name",var1[,var2…,varn] Reading from a data file.
A data record is read from the data file denoted by "name" and transferred into the specified variables.

CLOSE "name" Closing a data file.
The data file denoted by "name" is closed.

3. The individual DOS commands

General Instructions

INIT - Prepare a floppy disk

Syntax: **INIT**

allowed as direct command and in program mode.

The INIT command prepares a floppy disk for storing programs or data; it will be "initialized".

This means that the basic structure shown in the "Recording structure" section will be produced on the diskette.

40 tracks with 16 sectors each are set up and all address and data marks are written.

After writing, each sector is individually addressed and read.

The entire initialization process takes about 2-3 minutes.

After correct implementation, BASIC responds with READY and the next command can be entered.

The initialization process can be aborted at any time by pressing the BREAK button.

Warning:

With the INIT command, a non-write-protected diskette is overwritten without any further checking, i.e. any data on it is lost.

Possible Errors:

?DISK WRITE PROTECTED	The disk's write-protect notch is taped over.
?DISK I/O ERROR	An error occurred during the check read. (faulty disk or bad centering - see Insertion)

DRIVE - Drive selection.

Syntax: **DRIVE n**
n = drive number (1 or 2)

Allowed as direct command and in program mode.

The DRIVE command is used to select one of the two drives that can be connected.

After switching on the computer and after each copy command (DCOPY), drive 1 is automatically selected.

If you want to access drive 2, you must first switch to it with **DRIVE 2**.

All DOS commands, except DCOPY, are executed on the selected drive. Therefore, make sure that you have always selected the correct drive. An "INIT" command, e.g. on the wrong drive, inevitably leads to the destruction of a floppy disk with important data that happens to be there.

If you are not sure which drive is currently selected, execute a corresponding DRIVE command (DRIVE 1 or DRIVE 2) to be safe.

The DRIVE command. only changes the DOS internal pointers, a floppy disk access does not take place.

Possible Errors:

?FUNCTION CODE ERROR	Wrong drive selection (not 1 or 2)

DCOPY - Copy disk.

Syntax: **DCOPY**

Allowed only as a direct command.

The DCOPY command without any further parameters results in a complete copy of a floppy disk onto a second initialized floppy disk.

Copying is possible with one or two drives. With only one drive, however, you will have to change the diskettes several times during the copying process.

After entering the command, you will first be prompted to select the source and target drives.

SOURCE DISK (1/2)?

DESTINATION DISK (1/2)?

Answer each of these questions by pressing the "1" or “2” key.

Only own one drive; so answer "1" to each question.

Command execution can be aborted with CTRL/BREAK.

After the drive has been selected, the copying process begins. The entire RAM memory is used for this in order to have to switch between the source and target drive as little as possible.

If you copy from one drive to a second, the entire copying process runs automatically. If there is only one drive (from 1 to 1 or from 2 to 2), you will have the opportunity to insert the correct diskette before each read or write operation.

INSERT SOURCE DISKETTE
(PRESS SPACE WHEN READY)

before each reading from the source diskette, or

INSERT DESTINATION DISKETTE
(PRESS SPACE WHEN READY)

before each write to the target disk.

You can interrupt the copying process at any time by pressing the BREAK key.

The completion of the copying process is indicated with **READY**.

Warning:

- Note that the target disk must first be initialized.
- Data on the target diskette will be overwritten (ensure the correct drive and diskette selection).
- The entire available RAM area is overwritten by DCOPY, i.e. data or programs located there must first be saved or then reloaded.
- When using "Extended BASIC" the computer has to be re-initialized (switch off/on).
- After completion, drive 1 is always selected, regardless of a previous DRIVE command.

.

Possible Errors:

?ILLEGAL DIRECT	An attempt was made to call the DCOPY command from a program.
?DISK WRITE PROTECTED	The target disk's write-protect notch is taped over.
?DISK I/O ERROR	Write or read error on one of the two disks. (defective or bad / centering)

Note:

This is one of the most important DOS commands.

As already mentioned at the beginning, no floppy disk is a reliable data storage device in the long run (abrasion).

So make a copy of every diskette that contains programs and data that are important to you

- after the initial creation or acquisition
- after any significant change in content.

STATUS - Display the diskette status

(only from DISK BASIC V 1.2)

Syntax: **STATUS**

Allowed as direct command and in program mode.

The STATUS command determines and displays the space still available on the diskette.

The output comes in two forms. The first line shows the number of free sectors in the form:

nn RECORDS FREE

In the second line, the free bytes are specified in the form:

nn.nnn K BYTES FREE

Example:

STATUS
80 RECORDS FREE
10.0 K BYTES FREE

Possible Errors:

?DISK I/O ERROR	The occupancy overview of the diskette could not be read correctly.

File management functions

DIR - Display of the table of contents

Syntax: **DIR**

Allowed as direct command and in program mode.

The DIR command displays a directory of all programs and files stored on the diskette on the screen.

The listing includes file type and file name.

Possible file types:

T = BASIC - program (text file)
B = machine program (binary file)
D = data file

Example:

```
DIR
B:SCHACH
B:KALACH
T:AB.LAND
T:ANSCHR
D:KARTEI
READY
```

The disk contains:

- two machine programs, SCHACH and KALAH,
- two BASIC programs, AB.LAND and ANSCHR and
- a data file called KARTEI.

The listing can be stopped by pressing the space bar (SPACE) and continued with the same key.

Possible Errors:

?DISK I/O ERROR	The table of contents of the diskette could not be read properly.

SAVE - Saving a BASIC program to floppy disk

Syntax: **SAVE "name"**
"name" - program name with a maximum of 8 characters, enclosed in quotation marks.

Allowed as direct command and in program mode.

A BASIC program in memory is saved on the floppy disk under the file name "name".

The program is given the type designation "T" (text file).

In direct mode, the completion of the storage process is indicated with READY.

In program mode, the program is continued with the command following "SAVE".

Example:

SAVE “KARTEI”

transfers a BASIC program in memory to the floppy disk under the name "KARTEI".

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● No end of line (RETURN) or command separator ":" after the file name.
?DISK WRITE PROTECTED	The disk's write-protect notch is taped over.
?FILE ALREADY EXISTS	A file with the same name already exists on the diskette.
?DIRECTORY FULL	There is no more space in the table of contents (maximum 120 entries).

?DISK FULL	There are not enough free sectors on the diskette for the program.
?DISK I/O ERROR	An error occurred while writing or reading the floppy disk.

The writing process can be aborted at any time by pressing the BREAK key. However, depending on when the key is pressed, the entry in the table of contents is not always deleted (error in DOS).
In order to ensure problem-free diskette management, you should therefore check the table of contents with DIR in such a case and, if necessary, delete the file manually with ERA.

LOAD - Loading a BASIC program from diskette

Syntax: **LOAD “name”**

“name” - program name with a maximum of 8 characters, enclosed in quotation marks.

Allowed as direct command and in program mode.

A BASIC program saved on the diskette with the file name "name" is loaded into memory.

The completion of the storage process is indicated with READY.

Example:

LOAD “KFZ”

Transfers the BASIC program KFZ from the diskette to the memory.

You can then look at a BASIC program loaded in this way with LIST and modify it if necessary.

Warning:

Before writing a modified program back to the diskette, you must either first delete the program on it with "ERA" or give the modified program a different name.

Example:

```
LOAD “XYZ”
>READY
LIST
…
…       modify
…
ERA “XYZ”
>READY
SAVE “XYZ”
```

After the program has been read in, direct mode (BASIC warm start) is always accessed, regardless of whether the call was made directly or from within a program.

The reading process can be aborted at any time by pressing the BREAK key.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● No end of line (RETURN) or command separator ":" after the file name.
?FILE NOT FOUND	No program with the specified name could be found on the diskette..

?FILE TYPE MISMATCH	A file with the same name was found on the diskette, but this is not a BASIC program (file type = T).
?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

RUN - Load and start a BASIC program

Syntax: **RUN “name”**

“name” - program name with a maximum of 8 characters, enclosed in quotation marks.

Allowed as direct command and in program mode.

A BASIC program saved under "name" on the diskette is loaded into memory and executed.

Example:

RUN “GRAFIK”

The BASIC program "GRAFIK" is loaded and executed.

.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● No end of line (RETURN) or command separator ":" after the file name.

?FILE NOT FOUND	No program with the specified name could be found on the diskette..
?FILE TYPE MISMATCH	A file with the same name was found on the diskette, but this is not a BASIC program (file type = T).
?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

BSAVE - Saving a machine program on diskette

Syntax: **BSAVE “name”,aaaa,eeee**

“name” - program name with a maximum of 8 characters, enclosed in quotation marks.

aaaa - Program start address, 4 digits; in hexadecimal notation.

eeee - Program end address, 4 digits; in hexadecimal notation.

Allowed as direct command and in program mode.

A machine program in memory is written to the floppy disk from address "aaaa" to address "eeee" with the file name "name".

It receives the type designation "B" (binary file) in the table of contents.

In direct mode, the completion of the storage process is indicated with READY. In program mode, the program is continued with the command following BSAVE.

Instead of a machine program, this command can also be used to transfer any memory area to the diskette and then load it again with BLOAD.

Only BRUN requires an executable machine program as this is started immediately after loading.

Example:

BSAVE “BOWLING”,8000,94FF

The "BOWLING" machine program is transferred to the diskette from address 8000H to address 94FFH.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● Start and/or end address missing ● Start or end address not 4 digits hexadecimal (0~F) ● parameters not separated by comma,
?DISK WRITE PROTECTED	The disk's write-protect notch is taped over.
?FILE ALREADY EXISTS	A file with the same name already exists on the diskette.
?DIRECTORY FULL	There is no more space in the table of contents (maximum 128 entries).
?DISK FULL	There are not enough free sectors on the diskette for the program.
?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

The writing process can be aborted at any time by pressing the BREAK button. However, depending on when the key is pressed, the entry in the table of contents is not always deleted (error in DOS).
In order to ensure problem-free diskette management, you should therefore check the table of contents with DIR in such a case and, if necessary, delete the file manually with ERA.

BLOAD - Loading a machine program from diskette

Syntax: **BLOAD “name”**

“name” - program name with a maximum of 8 characters, enclosed in quotation marks.

Allowed as direct command and in program mode.

A machine program stored on the diskette with the file name "name" is loaded into the memory.

With a direct command, the end of the loading process is indicated with READY, in program mode the program is continued with the command following BLOAD.

Example:

BLOAD “UPR01”

Machine program UPR01 is loaded from the diskette.

The command is particularly suitable for loading machine program routines saved with BSAVE from a BASIC program and calling them as subroutines via USR.

Example:

```
…
…
220 BLOAD “UPR01”: ‘LOAD SUBPROGRAM
230 POKE 30862,0: ‘LSB START ADDRESS = 00
240 POKE 30863,176: ‘MSB START ADDRESS = B0
250 A = USR(0): ‘CALL SUBROUTINE
…
…
```

The subprogram UPR01 is to be loaded from diskette and called at address B000H.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● No end of line (RETURN) or command separator ":" after the file name.
?FILE NOT FOUND	No program with the specified name could be found on the diskette..
?FILE TYPE MISMATCH	A file with the same name was found on the diskette, but this is not a machine program (file type = B).
?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

BRUN - Loading and starting a machine program

Syntax: **BRUN “name”**

“name” - program name with a maximum of 8 characters,
enclosed in quotation marks.

Allowed as direct command and in program mode.

A machine program stored on the floppy disk under the file name "name" is loaded into memory and executed.

The program starts exclusively at the program start address (see BSAVE),
.

Example:

BRUN “FIFFI”

The "FIFFI" machine program is loaded and started.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● No end of line (RETURN) or command separator ":" after the file name.
?FILE NOT FOUND	No program with the specified name could be found on the diskette..
?FILE TYPE MISMATCH	A file with the same name was found on the diskette, but this is not a machine program (file type = B).

?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

RENAME - Renaming files and programs

Syntax: **REN “name1”,”name2”**

“name1” - File/program name, old, max. 8 characters, enclosed in quotation marks.

.

“name2” - File/program name, new, max. 8 characters, enclosed in quotation marks.

Allowed as direct command and in program mode.

A program or file on the disk under the name "name1" is renamed "name2".

Example:

REN “OTTO”,”ANTON”

The "OTTO" file is renamed to "ANTON".

Possible Errors:

?SYNTAX ERROR	● "name1" and/or "name2" are missing. ● "name1" or "name2 not in quotes ● names not separated by commas

.

?DISK WRITE PROTECTED	The disk's write-protect notch is taped over.
?FILE NOT FOUND	The file named "name1" is not on the disk.
?FILE ALREADY EXISTS	The file named "name2" already exists on the diskette.
?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

DCOPY - Copy a program

Syntax: **DCOPY “name”**

“name” - program name with a maximum of 8 characters,
enclosed in quotation marks.

Only permitted as a direct command,.

The DCOPY command with specification of a program name causes this program to be copied from one diskette to another.

After entering the command, you will first be prompted to specify the source and target drives.

SOURCE DISK (1/2)?
DESTINATION DISK (1/2)?

Answer each of these two questions by pressing the '1' or '2' key.

If you only have one drive, answer '1' to each question.

You can abort command execution with CTRL/BREAK.

After selecting the drive, the copying process begins. The copying takes place by calling the LOAD and SAVE routines, as they are also used with LOAD and BLOAD, or with SAVE and BSAVE.

For this reason, it is not possible to copy a single data file (file type = D) with the DCOPY command, as this is structured differently.

If you are copying to only one drive (SOURCE DISK = DESTINATION DISK), you will be prompted before loading

INSERT SOURCE DISKETTE
(PRESS SPACE WHEN READY)

and before writing the prompt

INSERT DESTINATION DISKETTE
(PRESS SPACE WHEN READY)

If you have inserted the correct diskette, press the spacebar to continue the function.

You can interrupt the copying process at any time with the BREAK button. If you do this during the writing process, please note the information on SAVE and BSAVE.

When copying is complete, the message READY appears.

Example: ((system outputs are marked with '>')

>READY
DCOPY “EMIL”
>SOURCE DISK (1/2)?
1
>DESTINATION DISK (1/2)?
1
>INSERT SOURCE DISKETTE
>(PRESS SPACE WHEN READY)
spacebar
…
… loading process

…
>INSERT DESTINATION DISKETTE
>(PRESS SPACE WHEN READY)
spacebar
…
… saving process
…
>READY

The program to be copied overwrites its original memory area in RAM.

After copying is complete, drive 1 is always selected, regardless of a previous DRIVE command.

Possible Errors:

?ILLEGAL DIRECT	An attempt was made to call the DCOPY command from a program.
?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● No end of line (RETURN) or command separator ":" after the file name.
?FILE NOT FOUND	No program with the specified name could be found on the diskette..
?FILE TYPE MISMATCH	An attempt was made to copy a data file.
?DISK WRITE PROTECTED	The target disk's write-protect notch is taped over.
?FILE ALREADY EXISTS	A program named "name" already exists on the target disk.
?DIRECTORY FULL	The table of contents of the destination disk is full. The program can no longer be entered (max. 128 files/programs).
?DISK FULL	There is no more space on the destination disk.

?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

ERASE - Delete a file or program on the floppy disk.

Syntax: **ERA “name”**

“name” - File/program name, max. 8 characters,
enclosed in quotation marks.

.
Allowed as direct command and in program mode.

A program or data file designated by "name" is deleted from the diskette.

To do this, the entry in the table of contents is deleted and all sectors occupied by this file are released.

Example:

ERA “DAT1”

The file named "DAT1" will be deleted.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes
?DISK WRITE PROTECTED	The target disk's write-protect notch is taped over.
?FILE NOT FOUND	No program with the specified name could be found on the diskette..
?DISK I/O ERROR	An error occurred while reading from the floppy disk. (faulty disk or centering problems)

Storage and processing of data

File organization and access

The LASER-DOS allows you to save data on the diskette from a BASIC program and then process them again.

This data is stored in special data files with the type code "D".

The storage form offered is "sequential". Sequential means that the data in the file is stored one after the other, like on a cassette. Reading or writing data always begins at the beginning of the file, further read and write calls access the subsequent positions of the file.

In contrast to this is the "direct" type of access (random access), with which any data in a file can be accessed directly. Unfortunately, the LASER-DOS does not support this type of memory as standard. However, it can easily be reproduced with a little knowledge of assembler/machine language and the help routines described in the last chapter.

Sequential access is data flow oriented, i.e. the number of characters for a write or read process can vary. This is also referred to as data records of variable length, where a data record is the sum of the data elements that are written to or read from the diskette with a write or read call.

Sequential files represent the simplest form of data storage and retrieval. They are ideal for storing raw data without wasting a lot of space between each data element. Data is read back in the same order as it was written.

In order to be able to access a data file, it must first be opened. A special OPEN call is available for this purpose. With the opening you also specify the type of access, whether data should be written or read.

After completing the data manipulations, each data file should be marked with a CLOSE
call to be closed.

When editing data files, there are a few important points to keep in mind:

- If a file that does not exist is opened for writing, it is created anew and positioned at the start of the file.

- If an existing file is opened for writing, it is positioned at the end of the file, i.e. the file is extended with the following write calls.

- If you want to rewrite an existing file from the beginning, you must first delete it.

- After opening, reading always begins at the beginning of the file. If you are looking for data within a file, you must read over the preceding ones.

- To update a sequential file, read in the source file and write the updated data to a new file.

- When reading the file, the exact structure of the data record to be read must be known. This does not refer to the length of the sentence and the individual elements; However, you must know the number of elements and the format of each element (string, integer, etc.) and provide an appropriate receiving field for each element.

- Within the file, the data is stored exclusively in ASCII format. The individual elements are separated by commas. For example, the number 1.2345 takes up 8 bytes of storage, including a space for the sign at the beginning and another space at the end. The text "ROBERT MAIER" takes up 12 bytes on the disk.

- A data record should not be longer than 200 bytes, otherwise problems with the internal data structure of BASIC will occur when reading.

  The 200 bytes count
  - - the individual characters
  - - the commas as element separators
  - - for numbers, the sign and an additional space
  - - a final RETURN (CR) at the end of the sentence

- A maximum of two files can be opened at the same time, whereby the types of access can be the same or mixed.

Warning:
If your system reports DISK-BASIC V1.0, work with one file at a time to be on the safe side. The management of two open files is still incorrect there and can lead to significant data loss.

- The DOS does not tell you when the end of the file is reached when reading, you have to determine this yourself, for example by writing a specific end identifier as the last in the file.

Example of sequential output:

We want to store a table of English to metric conversion data.

English unit	Metric unit
1 Inch	2.54001cm
1 Mile	1.60935 km
1 Acre	4046.86 qm
1 Cubic Inch	0.01639 ltr
1 U.S. Gallon	3.785 ltr
1 Liquid Quart	0.9463 ltr
1 lbs	0.45359 kg

The data should be structured as follows on the diskette and entered in a data file called "ENG>MET":

"English unit -> Metric unit” , conversion factor

e.g. "IN->CM", 2.54001

The following program creates such a file.

```
10 OPEN “ENG>MET”,1
20 FOR I% = 1 TO 7
30 READ E$,F
40 PR# “ENG>MET”,E$,F
50 NEXT
60 CLOSE “ENG>MET”
70 DATA “IN->CM”,2.54001,”MI->KM”,1.60935,”ACRE->QKM”,4046.86E-6
```

```
80 DATA “CU.IN->LTR”,1.638716E-2,”GAL->LTR”,3.785
90 DATA “LIQ.QT->LTR”,0.9463,”LB->KG”,0.45359
100 END
```

Line 10 creates the file "ENG>MET" and opens it for writing.
In line 40, one data record is written to the file.
Line 50 closes the "ENG>MET” file again.

Example of sequential input:

The following program reads the "ENG>MET" file into two parallel matrices and then asks about conversion problems.

```
10 CLEAR 1000
20 DIM E$(6),F(6)
30 OPEN “ENG>MET”,0
40 FOR I% = 0 TO 6
50 IN# “ENG>MET”,E$(I%),F(I%)
60 NEXT
70 CLOSE “ENG>MET”
100 CLS: PRINT “ CONVERSION ENGLISH=>METRIC”
110 PRINT: FOR I%=0 TO 6
120 PRINT TAB(4); USING “(## )   %     %   “;I%,E$(I%)
130 NEXT
140 PRINT @320, "WHICH CONVERSION (0-6)";
150 INPUT W%: IF W% > 6 THEN 190
160 INPUT "ENGLISH VALUE" ;V
170 PRINT "THE METRIC VALUE IS" V*F(W%)
180 INPUT "CONTINUE WITH <RETURN>";X
190 GOTO 100
```

Line 30 opens the file for input. Reading begins at the beginning of the file.
In line 50, a data set with the elements E$ (unit) and F (factor) is read and distributed to the matrices.
Note that the variable list when reading in is the same as the write command in the previous program.
In line 70 the file is closed again.

Updating a file

If you want to add one or more records to an existing file, open this file for writing and simply enter additional data records with PR#, which will be appended to the existing database.

If you want to change data within a file, we recommend the following procedure (not with DISK BASIC V1.0).

1. Open the file to be edited for reading.
2. Open a second new file for writing
3. Read a record and edit the data
4. Write the record to the new file
5. Repeat points 3 and 4 to the end of the file
6. Close both files
7. Delete the source file
8. Rename the new file to the original file

With DISK BASIC V1.0, the only solution is to read the file to be processed completely in the memory, process it and write it completely into the new file. However, this limits the size of the file to the available memory,

OPEN - Open a file.

Syntax: **OPEN “name”,n**

“name” - File/program name, max. 8 characters,
enclosed in quotation marks.

n - type of access
0 - Read
1 - Write

.
Permitted only in program mode.

The OPEN command opens a data file (type = D) for writing or reading.

The OPEN command creates a file control block internally for each open file, which contains function codes and pointers.

Furthermore, the following is positioned on the data according to the access code:

- When reading, always at the beginning of the file
- When writing to a new file, to the beginning of the file
- When writing to an existing file at the end of the file,

Since there are only two file control blocks in the system, only two files can be open at a time. The type of access is irrelevant, both can be opened for writing, both for reading or one for reading and the second for writing (see restriction DISK BASIC V1.0 on the previous pages).

Example:

OPEN “TEST”,0

The "TEST' data file is opened for reading.

A data file can only be opened once at a time. Attempting to open the same file again results in an error message.

Since the file control blocks (FCB) are located outside the BASIC programs, a file remains open if the calling program was aborted before the CLOSE call due to an error or by pressing the BREAK key and is perhaps no longer in memory. Such a file can no longer be opened without further ado.

If it happen that a BASIC program is aborted without properly closing its files, you should do so with a direct command (**CLOSE "filename"**).

Possible Errors:

?ILLEGAL DIRECT	An attempt was made to execute the OPEN command in direct mode.

?SYNTAX ERROR	● one or both parameters are missing ● no comma as separator ● filename not in quotes ● access type not 0 or 1
?FILE ALREADY OPEN	File is already open, if necessary close it with the direct command "CLOSE".
?FILE TYPE MISMATCH	The file addressed in the OPEN command is not a data file
?FILE NOT FOUND	A file to be opened for reading does not exist on the diskette.
?DISK BUFFER FULL	Two files are already open and no more file control block is available.
?DISK I/O ERROR	An error occurred while reading from the floppy disk.

PR# - Writing records to a file.

Syntax: **PR# “name”,item list**

“name” - File/program name, max. 8 characters,
enclosed in quotation marks.

item list - List of variables and values to be written to the file.
The individual elements are to be separated by commas

Permitted only in program mode.

Assembles a data record from the values in the element list and causes it to be written to the data file.

This must first have been opened for writing with an OPEN command.

Example:

```
200 A1 = -40.456: B$ = “STRING-VALUE”
210 OPEN “TEST”,1
220 PR# “TEST”,A1,B$,”THE VAR’S”
230 CLOSE “TEST”
240 END
```

After opening the "TEST" file in line 210, a data record is compiled in line 220 and written to this file.

The data record contains the current values of Al and B$ and also the character string "THE VAR'S". The values can later be read in again with an IN# command.

It must be ensured that the element list of the IN# command is the same as that of the PR# command with regard to the number and type of elements.

The values represented by the item list should not exceed 200 characters in total. In addition to the values themselves, this also includes all separators (commas) between the values, in the case of numeric values the sign position and a trailing space and finally the end of data record identifier (CR).

The record in the previous example would be 31 characters long

```
-40,456 ,STRING VALUE, THAT'S IT
```

Unfortunately, when creating the element list, one often does not know exactly how large the individual variables will be at the time of storage. Then only careful estimation helps. Always stay on the safe side and, if in doubt, split your element list into several PR# commands.

Unfortunately, the PR# command does not notice when a data record is too long. This is simply written to the diskette in its entirety. Reading in with the IN# command then causes problems, whereby in the simplest case "only" data is lost.

Writing a record does not necessarily result in a physical write to the file. Only full sectors are written to the diskette. The data of the PR# command are collected in an internal buffer with size of a sector. Whenever the buffer is full, it is transferred to a free sector on the disk and a new free sector is then determined. This writing of a sector can be done in the middle of a PR# command; multiple PR# commands may also be required to fill a sector.

A data record to be written with PR# is written regardless of sector boundaries. The sector is also referred to as a physical unit, while a data record represents a logical unit.

Possible Errors:

?ILLEGAL DIRECT	An attempt was made to execute the PR# command in direct mode.
?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● no item in the list ● no comma as separator
?FILE NOT OPEN	File was not previously opened.
?ILLEGAL WRITE	The file has been opened for reading.
?DISK WRITE PROTECTED	The disk's write-protect notch is taped over.
?DISK FULL	No more free sectors could be found on the diskette.
?DISK I/O ERROR	An error occurred while reading or writing to the diskette.

Warning:

If one of these errors occurs, the program is terminated with the corresponding error message. Please note that this file was not closed afterwards, you should do this manually.

IN# - Reading records from a file.

Syntax: **IN# “name”,item list**

“name” - File/program name, max. 8 characters,
enclosed in quotation marks.

item list - List of variables and values to be written to the file.
The individual elements are to be separated by commas

Permitted only in program mode.

IN# reads a record from the specified file and assigns the elements of that record to the specified variables.

The file must first have been opened for reading with an OPEN command.

Example:

```
200 OPEN “TEST”,0
210 IN# “TEST”,X,A$,B$
220 CLOSE “TEST”
…
…
```

This example refers to the data set created in the example of the PR# command in the "TEST" file. The data stored there are assigned to the variables of the IN# command in sequence.

After executing line 210, the variables contain the following values:

X = -40.456
A$ = “STRING-VALUE”
B$ = “THE VAR’S”

The element list of the IN# command must correspond to that of the PR# command with regard to the number and type of variables. Likewise, the order must be observed for different types, the naming is irrelevant,

If records are read continuously from a file with IN#, it is difficult to recognize the end of the file at the right time. There is no special "END OF FILE" identifier for LASER-DOS.

There are various possible solutions:

- the number of records is known, they are counted with a counter in the reading program,
- a second small file contains the sentence counter for the main file.
- A short label consisting of only one alphanumeric character (e.g. PR# "name", "A") is written in front of each correct record.

  In the reading program, this identifier is first read before each reading of a data record (e.g., IN# “name”, A$) If the receiving string variable is then empty, the end of the file has been reached.

Possible Errors:

?ILLEGAL DIRECT	An attempt was made to execute the IR# command in direct mode.
?SYNTAX ERROR	● no file name specified ● Filename not in quotes ● no item in the list ● no comma as separator
?FILE NOT OPEN	File was not previously opened.
?ILLEGAL WRITE	The file has been opened for reading.
?ILLEGAL READ	The file was opened for writing.
?DISK I/O ERROR	An error occurred while reading or writing to the diskette.

?REDO	The type of one of the specified variables does not match the data read in from the diskette. The program continues to run, the variable remains empty.
?EXTRA IGNORED	In the variable list of the IN# command fewer variables are given than values are present in the data set, the numbered values are ignored, the program continues.
??	The variable list contains more variables than there are values in the data set. The frogram now expects the missing values to be entered via the keyboard.

Warning:

If one of these errors occurs (except REDO, EXTRA IGNORED and ??), the program is terminated after the corresponding message has been output. Please note that this file was not closed, you should do this manually.

CLOSE - Closing a data file.

Syntax: **CLOSE “name”**

“name” - File/program name, max. 8 characters,
enclosed in quotation marks.

Allowed as direct command and in program mode.

A previously processed data file is closed with the CLOSE command..

If a file is open for reading or an inactive file (i.e. the last file access was not to this file) or in direct mode, only the file control block (FCB = File Control Block) is released again. Disk access does not take place.

However, if the CLOSE command is given in program mode and the file to be closed is open for writing and is currently active, the last sector in the buffer is also written back to the diskette so that no data is lost.

It is good programmer practice to close any open file after use. However, it is essential for output files, unless you accept the possibility of data loss.

Example:

CLOSE “MAILBOX”

The "MAILBOX" data file is closed.

It is always necessary to close and reopen a file if you want to change the type of access (e.g. from writing to reading).

If the file to be closed is not open at all, i.e. there is no open file control block for this file, the CLOSE command is skipped without any error message. This is especially useful for closing all files used in a program prophylactically at the end without checking which ones are currently open.

Possible Errors:

?SYNTAX ERROR	● no file name specified ● Filename not in quotes
?DISK WRITE PROTECTED	The disk's write-protect notch is taped over.
?DISK I/O ERROR	An error occurred while reading or writing to the diskette.

4. Error Messages

Summarized list of possible DOS error messages and their probable causes.

?DIRECTORY FULL	An attempt was made to save a program or a file on the floppy disk whose directory already contains 120 entries.
?DISK BUFFER FULL	An attempt was made to open a file with OPEN, although two files are already open.
?DISK FULL	There is no more free sector on the floppy disk.
?DISK I/O ERROR	An error occurred while writing or reading. e.g. address stamp not found; checksum wrong etc.
?DISK WRITE PROTECTED	An attempt was made to write to a floppy disk with the write-protect notch taped over.
?FILE ALREADY EXISTS	A program to be stored on the diskette is already there.
?FILE ALREADY OPEN	An OPEN call was issued to a file that is already open.
?FILE NOT FOUND	A file addressed for reading or a program to be loaded is not present on the diskette.
?FILE NOT OPEN	An attempt was made to use IN# or PR# to change a file that has not previously been opened.

?FILE TYPE MISMATCH	An attempt was made to access a file with the wrong type. LOAD/RUN = file type not equal to "T" BLOAD/BRUN - file type not equal to "B" OPEN - file type not equal to "D" DCOPY - file type="D"
?ILLEGAL DIRECT	An attempt was made to use a DCOPY command in program mode, or an OPEN, IN#, or PR# command in direct mode.
?ILLEGAL READ	The IN# command was issued for a file that is open for writing.
?ILLEGAL WRITE	The PR# command was issued for a file that is open for reading.
?INSUFFICIENT MEMORY FOR DOS	An attempt was made to initialize the DOS system on a LASER 110 or VZ200 without memory expansion.

5. Programming tips

1. As already mentioned in the description of the last sections, there is the problem that when a program is aborted due to an error or the BREAK key, not all files are necessarily closed correctly.

   A restart of such a program after error correction or similar usually leads to the message "FILE ALREADY OPEN".

   You can now manually complete these files in direct mode if their names are known.

   However, this is not sufficient for new files to be created in a program. Such files must then also be deleted, otherwise the file will be updated when OPEN is repeated and you will have your data in the file more than once.

   In all these cases, the following procedure is recommended:

   At the end of all programs under development you define a block with CLOSE calls and possibly also delete calls for all files addressed in the program. If the program is interrupted as mentioned above, simply call this routine with RUN line number.

   Example:

   In a program, you edit the three files DAT1, DAT2 for reading, and DAT3 is recreated.

```
…
…      own program
…
…
4800 END

20000 CLOSE “DAT1”
20010 CLOSE “DAT2”
20020 CLOSE “DAT3”
20030 ERA “DAT3”
20040 END
```

If a program is interrupted, you can use **RUN 20000** to clean up your files and restart your program without any problems after correction.

2. It is often necessary that a file must be present on the diskette when the program is started (see program example "Address Directory"), although it does not contain any data afterwards.

   Apply a similar technique to chen by defining the following lines at the end of the actual program:

   ```
   …
   …      own program
   …
   6000 END

   10000 OPEN “MAILBOX”,1
   10010 CLOSE “MAILBOX”
   10020 END
   ```

   With **RUN 10000** you create an empty file 'MAILBOX' on the floppy disk.

3. A bottleneck of LASER-D0S is that the file names must be specified directly in the commands and cannot be replaced by variables.

   How can you still edit different files in one program?

   Knowledge of the BASIC program structure is required to understand the following solution.

   Here are the most important points:

   - The start address of a BASIC program can be found in memory locations 78A4H and 78A5H (30884 and 30885 decimal).

   - A BASIC line has the following structure:

     2 bytes - pointer to the next line
     2 bytes - line number
     n bytes - line text
     1 byte - line end identifier (X'00")

BASIC keywords contained in line text, apart from the DOS commands, are represented in the text as one-character "TOKENS".
The space inserted between the line number and the line text in a program listing is not part of the line.

If these conditions are taken into account, the example below can be easily understood and reproduced.

The main point of this example is that all file calls are located at the beginning of the program, so they can be counted more easily and are not shifted when the program is changed later.

Example:

After selection by the user, a program evaluates one of three possible files DAT1, DAT2 or DAT3.

```
10 GOTO 100
20 OPEN “DAT1”,0:RETURN
30 IN# “DAT1”,A$.B$,C:RETURN
40 CLOSE “DAT1”:RETURN

100 CLEAR 1000
110 A=PEEK(30885)*256+PEEK(30884)
120 CLS
130 INPUT “FILE VERSION (1-3)”;X$
140 IF X$<”1” OR X$>”3” THEN 120
150 POKE A+23,ASC(X$)
160 POKE A+42,ASC(X$)
170 POKE A+69,ASC(X$)
180 GOSUB 20
190 GOSUB 30
…
…     edit the data, if necessary several records
…     read with GOSUB 30
…
400 GOSUB 40
410 END
```

Line 10 jumps to the actual beginning of the program.

In lines 20, 30 and 40 the file calls are defined as individual subroutines.

The program start address is determined in line 110.

Lines 130 and 140 ask for the desired file version.

If correct, this is transferred to the file names of lines 20, 30 and 40 in lines 150, 160 and 170.

Lines 180, 190 and 400 indicate file processing by calling subroutines as an example.

6. Application example "Address Management"

The "address management" program shown on the following pages shows a typical application example for diskette processing with the LASER-DOS.

It allows the entry, storage and editing of up to 100 addresses.

The addresses are stored on the diskette in a "MAILBOX" file.

The processing procedure was selected with regard to the weaknesses of DOS BASIC 1.0 in such a way that when the program starts the file content is read completely into the memory and at the end of the processing it is written back completely to the diskette due to changes.

Operation of the program

The program is loaded and started with RUN "ANSCHR". Immediately after the start, the content of the "MAILROX" file is transferred to the program-internal matrices. This file must be on the diskette, otherwise the program is terminated after an error message is output.

After the loading process, the menu is displayed

(1) NEW ENTRY
(2) UPDATE ENTRY
{3) DELETE ENTRY
(4) READ ENTRY
{5) LIST SORTED
(6) EXIT PROGRAM

You select one of these functions by entering the corresponding number.

After completing functions 1 to 5, which in my opinion are self-explanatory, the program returns to the menu output.

After completing function 6, the program is terminated.

If the data content was changed during the program run, the addresses are sorted alphabetically by last name and first name, if necessary, before functions 5 and 6 are executed.

In function 6, the data are written back to the diskette when changes have been made.

When starting the program, it expects the MAILBOX file to be present. If you are using the program for the first time, no such data is present on the diskette. You can create an empty "MAILBOX" file on the diskette with the following procedure:

```
LOAD "ACCESS"
RUN 3000
RUN
```

This creates a file "MAILBOX" and then starts the actual main program.

The program Structure

The program has a modular structure, i.e. each function is implemented in a self-contained routine.

After starting, the "MAILBOX" file is first read in (lines 220 - 280).

Note that for this file a solution was chosen for the end identifier, in which a label with a short alphanum. text is saved. The label is read and evaluated in lines 240 and 250, the "real" record is read in line 280.

The individual program routines for address processing (functions 1-3) will not be discussed in detail. These have nothing to do directly with the DOS.

You can analyze the routines yourself if necessary.

Just a hint. A "SHELL" SORT procedure was used to sort the addresses (lines 2200 - 2390), which is somewhat more complicated in terms of structure, but considerably better in terms of runtime than the simple and usual "BUBBLE" SORT.

The data is written back to the diskette in lines 1800 to 1920.

For security reasons, the data is first written to a temporary "TEMP" file. The old "MAILBOX" file is then deleted and the temporary file is then renamed "MAILBOX". Dis has the advantage that if errors occur during writing (DISK FULL or similar), at least the old file is still available.

If you are interested in the program, simply type it in and save it on the floppy disk with SAVE "ANSCHR".

```
100 '************************************************
110 '*
120 '*          ADDRESS MANAGEMENT
130 '*
140 '************************************************
150 '
160 CLEAR 2000
170 HD$="      ADDRESS DIRECTORY"
180 DIM NN$(99),VN$(99),TI&(99),ST$(99),NR$(99),PL$(99),OT$(99)
200 'READ ADDRESS LIST
210 GOSUB 2450
220 OPEN "MAILBOX",0
238 FOR N=8 TO 100
240 IN# "MAILBOX",A$
250 IF A$ = "" THEN 280
260 IN# "MAILBOX", NN$(N),VN$(N),TI$(N),ST$(N),NR$(N),PL$(N),OT$(N)
270 NEXT N
280 CLOSE "MAILBOX"
300 'PRINT MENU
310 GOSUB 2400
320 PRINT
330 PRINT TAB(4);"(1) NEW ENTRY"
340 PRINT TAB(4);"(2) UPDATE ENTRY"
350 PRINT TAB(4);"(3) DELETE ENTRY"
360 PRINT TAB(4);"(4) READ ENTRY"
370 PRINT TAB(4);"(5) SORTED LIST"
380 PRINT TAB(4};"(6) EXIT PROGRAM"
390 GOSUB 2500
400 A$=INKEY$: IF A$ < "1" OR A$ > "6" THEN 400
410 IF A$ = "1" THEN 500
420 IF A$ = "2" THEN 700
430 IF A$ = "3" THEN 1200
440 IF A$ = "4" THEN 1380
450 IF A$ = "5" THEN 1500
460 IF 50 = 1 GOSUB 2200
470 IF MO = 1 60TO 1800
480 CLS: END
500 'NEW ENTRY
510 GOSUB 2400
520 IF N = 99 PRINT "FILE MAILBOX ALREADY FULL": GOTO 2450
530 INPUT "LAST NAME ";NN$: IF NN$ = "" THEN 510
540 INPUT "FIRST NAME ";VN$
550 INPUT "TITLE ";TI$
560 INPUT "STREET ";ST$
```

```
570 INPUT "HOUSE NUMBER ";NR
580 INPUT "ZIPCODE ";PL$
590 INPUT “LOCATION “;OT$
600 GOSUB 2000
610 IF GF = 1 THEN 2550
620 NN$(N)=NN$: VN$(N)=VN$: TI$(N)=TI$: ST$(N)=ST$: NR(N)=NO
630 PL$(N)=PL$: OT$(N)=OT$
640 N = N +1
650 MO = 1: SO = 1
660 PRINT: PRINT "ENTRY COMPLETED*
670 GOTO 2600
700 'UPDATE ENTRY
710 GOSUB 2100: GOSUB 2000
720 IF GF = 0 THEN 2700
730 GOSUB 2400
740 PRINT "1. LAST NAME: ";NN$(I)
750 PRINT "2. FIRST NAME: ";VN$(I)
768 PRINT “3. TITLE: “TI$(I)
770 PRINT "4. STREET: ";ST$(I)
780 PRINT "5, HOUSE NUMBER: “;NR(I)
790 PRINT "6. ZIPCODE : ";PL$(I)
800 PRINT "7. LOCATION: “;OT$(I)
810 GOSUB 2500
820 A$=INKEY$: IF A$ < "0” OR A$ > "7" THEN 820
830 IF A$ = "0" THEN 300
840 IF A$ > "1" THEN 890
850 INPUT "LAST NAME “;NN$
860 IF NN$ = "" THEN 730
870 IF NN$<>NN$(I) THEN NN$(I) = NN$: SO = 1: MO = 1
880 GOTO 730
890 IF A$ > "2" THEN 930
900 INPUT "FIRST NAME ";VN$
910 IF VN$<>VN$(I) THEN VN$(I)=VN$: SO = 1: MO = 1
920 GOTO 730
930 IF A$ > "3" THEN 970
940 INPUT "TITLE ";TI$
950 IF TI$<> TI$(I) THEN TI$(I)=TI$: MO = 1
960 GOTO 730
970 IF A$ > "4" THEN 1010
980 INPUT "STREET “;ST$
990 IF ST$<>ST$(I) THEN ST$(I)=ST$: M0 = 1
1000 GOTO 730
1010 IF A$ > "5" THEN 1050
1020 INPUT "HOUSE NUMBER ";NR
```

```
1030 IF NR<>NR(I) THEN NR(I)=NR: MO = 1
1040 GOTO 730
1050 IF A$ > "6" THEN 1090
1060 INPUT "ZIP CODE “;PL$
1070 IF PL$<>PL$(I) THEN PL$(I)=PL$: MO = 1
1080 GOTO 730
1090 INPUT “LOCATION “;OT$
1100 lF OT$<>OT$(I) THEN OT$(I)=OT$: MO = 1
1110 GOTO 730
1200 ‘DELETE ENTRY
1210 GOSUB 2100: GOSUB 2000
1220 IF GF = 0 THEN 2700
1230 PL$(I) = "XXXX"
1240 PRINT: PRINT "ENTRY DELETED"
1250 MO = 1: GOTO 2600
1300 ‘READ ENTRY
1305 IF SO = 1 GOSUB 2200
1310 GOSUB 2100: GOSUB 2000
1320 IF GF = 0 THEN 2700
1330 GOSUB 2400
1340 PRINT TI$(1)
1350 PRINT VN$(I)” “NN$(I)
1360 IF ST$(I) = "" THEN 1390
1370 PRINT ST$(1);
1380 IF NR(I) = 0 PRINT " " ELSE PRINT NR(I)
1390 IF PL$(I) = "" THEN 1395 ELSE PRINT PL$(I)”   ";
1395 PRINT OT$(I)
1400 I = I + 1: G0SUB 1610
1410 IF I < N THEN 1330 ELSE 300
1500 ‘OUTPUT LIST
1510 IF SO = 1 GOSUB 2200
1520 I=0
1530 GOSUB 2400
1540 IF N = 0 PRINT "NO ENTRIES EXIST”: GOTO 2600
1550 FOR J = 1 TO 12
1560 IF I = N THEN 1600
1570 IF PL$(I) = "XXXX* THEN 1590
1580 PRINT NN$(I)", "VN$(I)
1590 I= I + 1: NEXT J
1600 GOSUB 1610: IF I < N THEN 1530 ELSE 300
1610 PRINT @480, "<RETURN> = CONTINUE, <E> = EDN";
1620 A$ = INKEY$
1630 A$ = INKEY$: IF A$ = "” THEN 1630
1640 IF A$ = "E" THEN I = N: RETURN
```

```
1650 IF A$ <> CHR$(13) THEN 1630
1660 RETURN
1800 ‘WRITE DATA TO DISK
1810 GOSUB 2450
1820 OPEN "TEMP",1
1830 IF N = 0 THEM 1890
1840 FOR I = 0 TO N-1
1850 IF PL${I) = "XXXX* THEN 1880
1860 PR# "TEMP", "A"
1870 PR# “TEMP",NN$(I),VN$(I),TI$(I),,ST$(I),NR(I),PL$(I),OT$(I)
1880 NEXT I
1890 CLOSE "TEMP"
1900 ERA "MAILBOX"
1910 REN "TEMP", "MAILBOX"
1920 CLS: END
2000 ‘SEARCH ENTRY IN LIST
2010 FOR I = 0 TO N-1
2020 IF PL$(I) = "XXXX" THEN 2060
2030 IF NN$ <> NN$(I) THEN 2060
2040 IF VN$ = "" THEN 2070
2050 IF VN$ = VN$(I) THEN 2070
2060 NEXT I: GF = 0: RETURN
2070 GF = 1: RETURN
2100 ‘READ SEARCH CRITERIA
2110 GOSUB 2400
2120 INPUT "LAST NAME ";NN$
2130 IF NN$ = "" THEN 2110
2140 INPUT "FIRST NAME ";VN$
2150 RETURN
2200 ‘SORT THE ENTRIES
2210 GOSUR 2450: SO = 0
2220 IF N < 2 RETURN
2230 M = N
2240 M = INT (M/2): IF M = 0 RETURN
2250 J = 1: K= N-M
2260 I = J
2270 L = I + M
2280 X = I - 1: Y = L - 1
2290 IF NN$(X) < NN$(Y) THEN 2390
2300 IF NN$(X) > NN$(Y) THEN 2320
2310 IF VN$(X) <= VN$(Y) THEN 2390
2320 NN$=NN$(X): VN$=VN$(X): TI$=TI${(X): ST$=ST$(X): NR=NR(X)
2330 PL$=PL$(X):OT$=OT$(X):NN$(X)=NN$(Y):VN$(X)=VN$(Y)
2340 TI$(X)=TI$(Y):ST$(X)=ST$(Y):NR(X)=NR(Y)
```

```
2350 PL$(X)=PL$(Y): OT${X)=0T$(Y): NN$(Y)=NN$
2360 VN$(Y)=VN$: TI$(Y)=TI$: ST$(Y)=ST$: NR(Y)=NR
2370 PL$(Y)=PL$: OT$(Y)=OT$
2380 I = I - M: IF I > 0 THEN 2270
2390 J = J +1: IFJ >K THEN 2240 ELSE 2260
2400 ’HEADER OUT
2410 CLS: PRINT HD$: PRINT: RETURN
2420 ‘PLEASE WAIT
2460 CLS: PRINT @228,"***** PLEASE WAIT *****”
2470 RETURN
2500 ’READ DIGIT
2510 PRINT: PRINT "PLEASE ENTER NUMBER (0=END)": PRINT
2520 A$ = INKEY$
2530 RETURN
2550 ‘MESSAGE "ADDRESS EXISTS"
2560 PRINT: PRINT "ADDRESS ALREADY EXISTS"
2600 ‘WAIT FOR RETURN
2610 PRINT: INPUT "CONTINUE WITH <RETURN>";X
2620 GOTO 300
2700 ‘MESSAGE "ADDRESS NOT AVAILABLE"
2710 PRINT: PRINT "ADDRESS NOT AVAILABLE"
2720 GOTO 2600
3000 OPEN "MAILBOX”,1
3010 CLOSE “MAILBOX"
3020 END
```

7. Technical Information

Structure and organization of the diskette

Structure of the diskette after initialization

Before you can work with a floppy disk, it must have the basic structure of tracks and sectors.

This basic structure is written to the diskette using the initialization (INIT command).

It consists of 40 tracks with 16 sectors each, each with 128 bytes of data capacity. As noted in the "Recording Structure" section, each sector is followed by a certain "overhead" consisting of necessary synchronization and addressing fields. This results in a total length of 154 bytes per sector.

Such a sector has the following basic structure:

Bytes 8 - 6	Address synchronization	7 * X'80'
Bytes 7 - 10	Address mark	X'FE E7 18 C3'
Bytes 11 - 13	Address field	
	Byte 11 = track number	(0 - 39)
	Byte 12 = sector number	(0 - 15)
	Byte 13 = checksum "address field"	(Track# + Sector#)
Bytes 14 - 19	Data synchronization	6 * X'80'
Bytes 20 - 23	Data mark	X'C3 18 E7 FE'
Bytes 24 - 151	Data field	= 128 bytes
Bytes 152 - 153	checksum "data field"	

Each sector is written completely during initialization, with the data field and checksum (bytes 24 - 153) being set to X'00'.

The sectors are not numbered consecutively around the disk, but arranged in jumps of three (see Figure 1.6). This achieves the effect that consecutive sectors of a track can be reached during one revolution of the disk if a certain processing time in between is not exceeded.

This is 94 ms from the end of a sector until the beginning of the next sector appears in the numerical order under the read/write head. 94 ms is a huge amount of time, in which, computer can do extensive data manipulation.

Of the tracks on a floppy disk, 39 are available for storing programs and data,

The first track of a diskette (track 0) is used for diskette management. It contains the table of contents of the diskette and a sector allocation overview.

Table of Contents

The table of contents of the diskette is in the first 15 sectors of track 0 (sector 0 - 14).

Each entry occupies a space of 16 bytes,
This gives a capacity of 8 entries per sector and 8 x 15 = 120 entries in the whole directory (see error message "?DIRECTORY FULL").

An entry in the table of contents has the following structure:

Byte 0	occupancy status / file type 0 - end of used entries in the table of contents 1 - released entry (e.g. after a "ERA") D - entry refers to a data file T - entry refers to a text file (BASIC program) B - entry refers to a binary file (machine Program)
Byte 1	separator (always ':')

Bytes 2 - 9	file name
Byte 10 - 11	address of the first sector of this file Byte 10 - track number Byte 11 - sector number
Byte 12-13	only with file type = T or B Program start address in memory
Byte 14 - 15	only with file type = T or B Program end address in memory

With the "DIR" command, the first 10 bytes of each assigned entry are simply output on the screen without any preparation.

If a file is deleted, only the status byte (byte 0) is set to '1'. All other entries are retained.

The sector administration

The last sector of track 0 contains the allocation overview for the disk sectors.

A bit is reserved there for each sector from track 1, which indicates whether the corresponding sector is free (bit = 0) or occupied (bit = 1).

With 39 tracks and 16 sectors per track, this results in 624 required bits or 78 bytes containing relevant information in this sector.

When writing a file, this allocation overview is used to determine the sectors required for storage. The sectors are always occupied from front to back and any gaps that may have arisen are filled in by deleting them.

Mapping example:

Track 0 / Sector 15

Byte 0	⇒	Track 0, Sectors 0 - 7
Byte 1	⇒	Track 0, Sectors 8 - 15
Byte 2	⇒	Track 0, Sectors 0 - 7
…		
…		
…		
Byte 77	⇒	Track 39, Sectors 8 - 15

Storage of programs and files

All programs stored on the diskette receive a corresponding entry in the table of contents, with the type of file or program being noted in the first byte.

This type designation is the only difference between text files (BASIC programs) and binary files (machine programs). The recording structures are identical.

The different type identifiers result in different handling after loading or starting such a program (see the LOAD/RUN or BLOAD/BRUN command descriptions).

Bytes 10 and 11 of the table of contents contain a pointer to the first sector occupied by this program.

Bytes 12 - 15 of the table of contents contain information about the memory area to which this program is to be transferred when loading. Bytes 12 and 13 contain the start address and bytes 14 and 15 the end address of the transfer area.

The data sectors contain in bytes 0 - 125 of the data field a 1:1 Kopie of the memory area, i.e. in binary data representation,

The sectors occupied by a program do not have to be physically consecutive, but can be scattered on the diskette. In order to still be able to read a program in one go, the individual sectors are indexed one below the other.

In bytes 126 and 127 of the sector there is a pointer to the next sector of this program (track and sector number) or '0' in the last occupied sector.

Data files with the type designation 'D' do not contain information about a memory area to be occupied in the table of contents, bytes 12 - 15 are irrelevant.

Bytes 10 and 11 also contain the pointer to the first occupied sector.

As with the programs, the individual sectors of the file are linked to one another.

Each sector of a data file contains the actual data in the first 126 bytes of the sector and a pointer to the next occupied sector or '0' at the end of the file in the last two bytes.

The structure of the data in the first 126 bytes differs from the other two file types.

Data representation is in ASCII format only. Storage is based on data records, with each PR# command writing a complete data record to the file.

Records contain a defined end identifier. This is the ASCII character for "Carriage Return" X'0D'.

A data record is not based on sector boundaries. There can be several records in one sector; a data record can also extend over several sectors. Except for the first record of a file, records do not have to start on a sector boundary either.

Within the data records, the various data fields are separated from one another by commas; they are assigned to the variables defined in the IN# command when they are read.
-

Memory resident workspaces

To process the diskettes, DOS creates various data structures in the last 310 bytes of the available RAM memory, which contain processing vectors, file management blocks and input/output buffers (Figure 1.7).

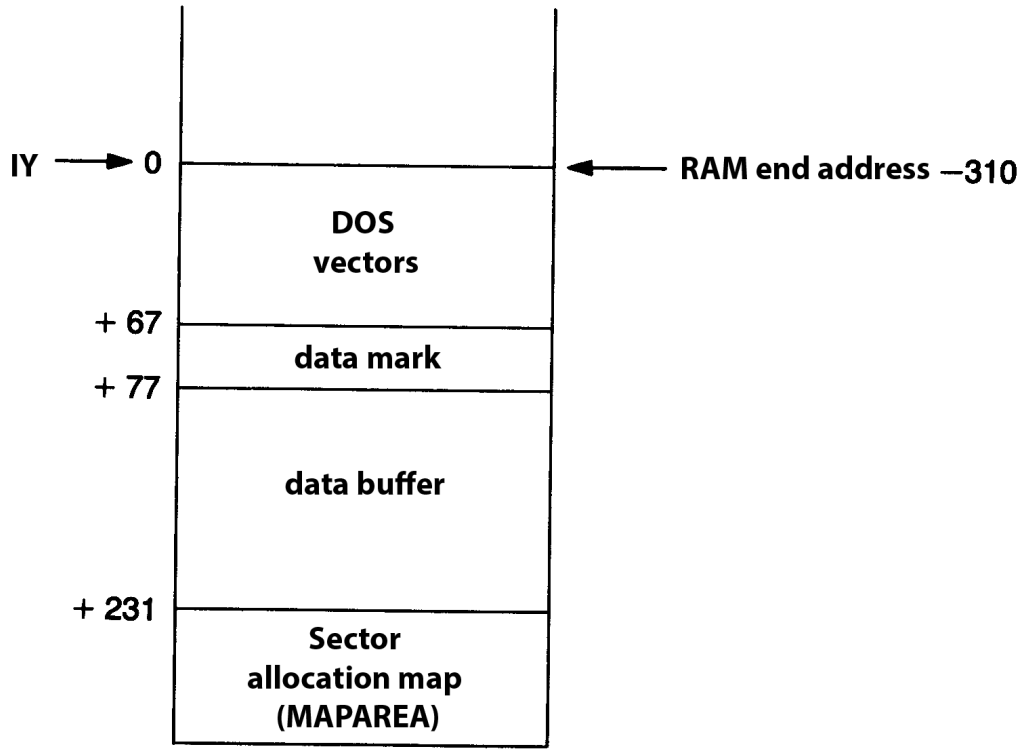


Figure 7.1 The memory areas of DOS

DOS vectors

The first 67 bytes of this DOS work area contain the DOS vectors.

The address of the start of the DOS vectors is stored by DOS during system initialization to the Z80 register 'IY'. DOS expects that this register will not be changed by user programs, otherwise a system crash will inevitably occur and data on the diskette may also be destroyed (bitter experience of the author).

The DOS vector area is structured as follows:

DOSVTR = IY

Name	Bytes	Offset	Description
FILNO	1	IY+0	File number. When processing a data file, this is the number of the file management block (FCB) used. 0 = FCB1 , 1 = FCB2
FNAM	8	IY+1	Filename Name of the file to edit. must be entered by the user program before each file/program access.
TYPE	2	IY+9	File Type Byte 1 = target type. Byte 2 = actual type. From the user program, the type of file to be processed is in the first byte.
DK	1	IY+11	Selected Drive. X'10' = Drive 1 X'80' = Drive 2 X'10' is set during initialization.
RQST	1	IY+12	Access type. 0 = read 1 = write Must be set by the user program. With BASIC, this is done with the OPEN command.
SOURCE	1	IY+13	Starting drive (source) used by DCOPY command (1 or 2)
UBFR	2	IY+14	Address of a user buffer area to or from which data is to be transferred. When loading and saving programs, this is the program area. When reading data files, it is the

			BASIC input/output buffer
DESTIN	1	IY+16	Target drive for the DCOPY command (1 or 2)
SCTR	1	IY+17	Number of the Sector to be addressed
TRCK	1	IY+18	Number of the Track to be addressed,
RETRY	1	IY+19	Retry counter for read errors (checksum). Set to 10 upon initialization.
DTRCK	1	IY+20	Current track number over which the read/write head is located.
NSCT	1	IY+21	Marker field for the next sector to be addressed.
NTRK	1	IY+22	Marker field for the next track to be addressed.
FCB1	13	IY+23	File Control Block 1. (see Structure description)
FCB2	13	IY+36	File Control Block 2. (see Structure description)
DBFR	2	IY+49	Pointer to the DOS data buffer for writing and reading a sector This is located immediately after the DOS vectors in the work area.
LTHCPY	1	IY+51	Copy of the command byte sent to the Floppy Disk Controller.
MAPADR	2	IY+52	Pointer to the DOS buffer in which the sector occupancy overview is temporarily stored. This is behind the data buffer in the

			work area.
TRKCNT	1	IY+54	Track counter for the DCOPY command
TRKPTR	1	IY+55	Track pointer for the DCOPY command.
PHASE	1	IY+56	Step pulse raster for track adjustment.
DCPYF	1	IY+57	Flag for DCOPY
RESVE	10	IY+58	reserved for extensions.

File Control Blocks (FCB)

Within the DOS vectors are two 13 byte file control blocks, FCB1 and FCB2.

These are required when processing data files in order to keep status and control information about the file being accessed.

A free file control block is determined by the OPEN command and provided with the necessary parameters for the file to be opened.

The IN# and PR# commands are based on the relevant file control block, e.g. which sector of the file is to be read and at which byte of this sector processing is to be continued.

The file control blocks are released again by the CLOSE command.

Since there are only two of these blocks, only two files can be open at the same time.

The File Control Block has the following structure:

FCB1 or FCB2

Name	Bytes	Description
FLAG	1	Indicates the status of the FCB. 0 - FCB not used 1 - FCB used, file currently not active 2 - FCR used, file active. Active means that a current sector of this file is in the data buffer for processing
ACCESS	1	Access type for this file. 0 - read 1 - write
FNAM	8	Filename
TRK#	1	Track Number
SCTR#	1	Sector Number of currently processed Sector
PTR	1	Pointer to the next byte to be processed in the above sector.

Input/Output Buffer

In the DOS work area there are two buffer areas, one for temporary storage of the sectors to be read or written and a second for the sector occupancy overview.

Data Buffer (DBFR)

This buffer has a size of 154 bytes and serves as a buffer for direct data exchange with the floppy disk.

When writing, the sectors are transferred from the data buffer to the diskette; when reading, the sectors are transferred from the diskette to the data buffer.

During initialization, the 10 bytes of the data mark are set in front of the data buffer, so that a complete information block (data mark + data field) is available when a sector is written.

During normal read/write operations, only the first 128 bytes of the data buffer are used to hold a sector's data field.

The full length of 154 bytes is only required during diskette initialization to accommodate a complete sector, including all sync fields, address fields, and identifiers.

Sector Occupancy Overview (MAP)

At the end of the DOS work area there is an 80-byte buffer area in which the sector occupancy overview from sector 15 of track 0 on the diskette is buffered.

When saving a program or writing a data file, the sectors are selected and allocated exclusively in this buffer area after the current sector has been read in at the beginning. Only when the saving process for the program is complete, the assignment overview (MAP) is written back to the diskette.

8. Communication between the DOS and the Floppy Disk Controller

The connection between the DOS and the floppy disk controller is established via 4 input/output ports. These are ports 10H in hexadecimal notation. 11H, 12H and 13H.

PORT 10H = command register (O/P LATCH)

The control information is transferred to the command register of the floppy disk controller via this port.

Bit 7 = Drive 2 - select (1 = yes)

Bit 6 = Access type (0 = write, 1 = read)

Bit 5 = Output pulse when writing to diskette

Bit 4 = Drive 1 - select (1 = yes)

Bits 3-0 = Step phases for disk head adjustment

A copy of the port contents is kept in the DOS vectors in the LTHCPY field.

This field is initialized with '0110 0001'.

The current step phases are held and processed in the "PHASE" field of the DOS vectors.

The selected drive is in the DK field.

When a drive is powered on, drive select (DK) and step phase (PHASE) are linked to the contents of LTHCPY.

PORT 11H = READ and STROBE SHIFT register

The data is read from the diskette via this port. They are inserted serially from the left bit by bit by the floppy disk control.

PORT 12H = POLL DATA

This port is used for synchronization when reading.

A negative pulse is generated when the next bit is available at port 11.

PORT 13H = READ/WRITE PROTECT STATUS

The write protection status of a floppy disk can be determined via this port (write protection notch taped over or not).

The result is passed in bit 7.
(1 = read only)

9. The most important DOS routines and their application iп machine programs.

Call and Overview

As already mentioned several times, the DOS occupies the address space 4000H to 5FFFH. It is in ROM chips built into the floppy disk controller. The floppy disk controller is connected to the system bus of the computer.

The presence of a floppy disk controller is determined when the computer is switched on by checking a specific byte sequence (AA 55 E7 18). If this byte sequence is found, the subsequent initialization routine is called automatically, which among other things attaches a BASIC vector so that the special DOS commands can be recognized and executed.

This is the RESTART 10 vector at address 7803H in the BASIC communications area. If this vector is called up by BASIC within the command analysis (at address 1D5AH), the DOS first checks whether a special DOS command is present. If no, the program continues with the normal BASIC command execution. If a DOS command is recognized, the necessary execution routines are called in DOS.

The DOS vector area at the end of the memory is also built up by the DOS initialization routine and filled with initial values.

You can also check within a machine program whether a floppy disk controller is connected to the system bus by checking the corresponding byte sequence at address 4000H.

The DOS itself consists of a number of self-contained routines. A large part of it can also be called from machine programs, so that individual diskette and data handling can be programmed and executed there.

You could use it to edit the existing file system of DOS, but you could also create completely new structures and forms of processing, such as the previously mentioned files with direct access, which are not supported by DOS.

Jumping to the individual routines directly at their start addresses would be one of the possible uses. However, you would then fix the programs to a D0S version, since each time the DOS changes, some of these addresses are also shifted.

A more elegant solution is to use a jump table at the beginning of the DOS, which was created by the manufacturer for this purpose and allows the most important subroutines to be called.

It is called using the Z80 command

CALL xxxxH

The following subroutines can be reached via this jump table:

Name	Call	Function
PWRON	**CALL 4008H**	Turn on the drive
PWROFF	**CALL 400BH**	Turn off the drive
ERROR	**CALL 400EH**	DOS error handling
RDMAP	**CALL 4011H**	Load sector occupancy Map
CLEAR	**CALL 4014H**	Delete sector
SVMAP	**CALL 4017H**	Write sector occupancy Map
INIT	**CALL 401AH**	Initialize disk
CSI	**CALL 401DH**	Interpret command parameters
HEX	**CALL 4020H**	Conversion ASCII to HEX
IDAM	**CALL 4023H**	Look for the address mark on the diskette
CREATE	**CALL 4026H**	Write an entry in the table of contents
MAP	**CALL 4029H**	Detect a free sector
SEARCH	**CALL 402CH**	Find file in table of contents

FIND	**CALL 402FH**	Look for free space in the table of contents
WRITE	**CALL 4032H**	Write sector to disk
READ	**CALL 4035H**	Read sector from disk
DLY	**CALL 4038H**	n milliseconds delay
STPIN	**CALL 403BH**	Advance head n tracks inward
STPOUT	**CALL 403EH**	Advance head n tracks outward
LOAD	**CALL 4041H**	Load a program
SAVE	**CALL 4044H**	Save a program

However, before you call one of these subroutines, very specific input parameters often have to be set, e.g. entry of the file name in the DOS vector, drive selection in the DK field, etc...

It is also important to know which results such a subroutine returns where and which possible error codes are reported.

You should also know which registers are changed in the subroutines
so you can save them beforehand.

The following pages describe each of these subroutines with their input and output values, registers used and error codes. In addition, there is a function description and an application/call example.

Two additional functions that you cannot access via the jump table but can easily be programmed yourself are also listed. These are the drive selection and checking the write protection of the floppy disk.

Be careful not to change register IY in your program. The start address of the DOS vectors is entered there during the initialization, which not only the DOS, but also you constantly need when using the routines mentioned above.

Some routines return error codes in register A. In any case, you should check whether your call was successful or not.

All data that is moved between the computer and the floppy disk uses the data buffer in the DOS work area as temporary storage. Remember that each time a sector is read or written, its content is modified.

The operating system generates an interrupt every 20 ms, which is normally used to update the screen content and evaluate the keyboard.

However, these interruptions are not desirable for disk accesses, where very precise time intervals are important; they would make error-free access impossible.

For this reason you must switch off the interrupts with DI (disable interrupts) and then switch them on again with EI (enable interrupts) before each diskette access.

In many cases, you must also check whether the diskette to be edited is write-protected; otherwise, write operations are still performed.

PWRON - Turn on a drive

Call: **CALL 4008H**

Input: DK (IY+11) in the DOS vectors = Drive identifier
X'10' = Drive 1
X'80' = Drive 2

Exit: -

Registers used: A

Error handling: none

The drive selected in DK will be powered on. The drive motor runs and the red LED on the front of the drive lights up.

You should wait a few milliseconds for the rotation speed to stabilize before accessing this drive.

Example:

```
…
…
DI                ; disable interrupts
LD    A,80H       ; select Drive 2
LD    (IY+11),A
CALL  4008H       ; turn on the drive
LD    BC,50       ; 50 ms delay
CALL  4038H
…
…
```

Drive 2 turns on, then the program waits 50ms for stabilization.

PWROFF - Turn off a drive

Call: **CALL 400BH**

Input: -

Exit: -

Registers used: A

Error handling: none

A powered-on drive is powered off. The drive motor stops and the LED on the front of the drive goes out.

Example:

```
…
CALL  400BH       ; turn off the drive
…
```

ERROR - Error handling

Call: **CALL 400EH**
or: **JP 400EH**

Input: A = Error code (0 - 17)

Exit: Jump to the BASIC interpreter

Registers used: AF, BC. DE, HL

Error handling: none

A powered-on drive is powered off. The drive motor stops and the LED on the front of the drive goes out.

An error message is output according to the error code transferred in the A register. A possibly switched on drive is switched off (with error code > 1).

This routine differs from the other routines insofar as it jumps to the BASIC interpreter instead of to the calling program.

The stack pointer is set to the BASIC stack.

A=	generated message	further process
0	none	to the BASIC interpreter (1B9AH)
1	?SYNTAX ERROR	Release and further expiry about 1997H
2	?FILE ALREADY EXISTS	to the BASIC interpreter (1B9AH)
3	?DIRECTORY FULL	to the BASIC interpreter (1B9AH)
4	?DISK WRITE PROTECTED	to the BASIC interpreter (1B9AH)
5	?FILE NOT OPEN	to the BASIC interpreter (1B9AH)
6	?DISK I/O ERROR	to the BASIC interpreter (1B9AH)
7	?DISK FULL	to the BASIC interpreter (1B9AH)

8	?FILE ALREADY OPEN	to the BASIC interpreter (1B9AH)
9	?SECTOR NOT FOUND	to the BASIC interpreter (1B9AH)
10	?CHECKSUM ERROR	to the BASIC interpreter (1B9AH)
11	?UNSUPPORTED DEVICE	to the BASIC interpreter (1B9AH)
12	?FILE TYPE MISMATCH	to the BASIC interpreter (1B9AH)
13	?FILE NOT FOUND	to the BASIC interpreter (1B9AH)
14	?DISK BUFFER FULL	to the BASIC interpreter (1B9AH)
15	?ILLEGAL READ	to the BASIC interpreter (1B9AH)
16	?ILLEGAL WRITE	to the BASIC interpreter (1B9AH)
17	?BREAK	to the BREAK routine (1DA0H) in BASIC

Example:

```
…
…
LD     A,7              ; Error code 7
CALL 400EH              ; Output message "DISK FULL"
```

The message "?DISK FULL" is output and then BASIC responds with READY.

Note:

Using the line number field in the BASIC communication area (78A2H), the ERROR routine distinguishes whether it is a direct command or a program command.

If field 78A2H/78A3H contains a value not equal to X'FFFF', this is interpreted as a line number and this is output after the error message (error codes 1-16).

This function can perhaps also be useful when testing machine programs by setting specific values in 78A2H/78A3H which, if an error occurs, give you an indication of the corresponding point in the program.

Example:

```
        …
        …
        OR    A          ; check if error occurred
        JR    Z,XY       ; no, go on!
        LD    HL,10      ; set row identifier
        JP    400EH      ; call error routine
XY      …
        …
```

If A contains a value not equal to 0, the corresponding error message is output with information about the location of the occurrence.
e.g. A= 3, then "?DIRECTORY FULL IN 10".

RDMAP - Load allocation Map

Call: **CALL 4011H**

Input: The corresponding drive must be switched on.

Exit: The sector allocation map is located in the 80-byte buffer at the end of the DOS work area (MAPAREA).

The drive remains powered on.

Registers used: AF, BC. DE, HL

Error handling: In the event of an error, this routine automatically branches to the ERROR routine. Custom error handling is not possible.

The sectors - occupancy overview (occupancy map) is loaded from sector 15 of track 0 from the diskette into the main memory and transferred to the MAPAREA at the end of the DOS work area.

Note that you are responsible for turning the drive on and off yourself.

Example:

```
…
…
DI                  ; disable interrupts
LD   A,10H          ; select Drive 1
LD   (IY+11),A
CALL 4008H          ; and turn on
LD   BC,50
CALL 4038H          ; 50 ms delay
CALL 4011H          ; load allocation map
CALL 400BH          ; turn off the drive
LD   L,(IY+52)      ; start address of allocation map ..
LD   H,(IY+53)      ; .. into HL
EI                  ; enable interrupts again
…
…
```

The sector occupancy overview is loaded from the floppy disk in Drive 1. The start address of the MAPAREA is then made available in the register pair HL.

Internally called routines: READ

CLEAR - Deleting a sector on the floppy disk

Call: **CALL 4014H**

Input: The corresponding drive must be switched on.
SCTR (IY+17) = Sector Number
TRCK (IY+18) = Track Number

Exit: The addressed sector is physically erased from the disk..

Registers used: AF, BC. DE, HL

Error handling: In the event of an error, this routine automatically branches to the ERROR routine.Custom error handling is not possible.

The sector addressed in the DOS vectors SCTR (IY+17) and TRCK (IY+18) is physically erased on the diskette, i.e. overwritten with binary zeros (X'00').

Note that you have to take care of switching the drive on and off yourself.

Before erasing the sector, make sure the disk is not write-protected.

Example:

```
…
…
DI                  ; disable interrupts
LD    A,80H         ; select Drive 2
LD    (IY+11),A
CALL  4008H         ; and turn on
LD    BC,50         ; 50 ms delay
CALL  4038H
IN    A,(13H)       ; check write protection
OR    A
LD    A,4
JP    M,400EH       ; read only!
LD    (IY+17),12    ; set Sector Number
LD    (IY+18),28    ; set Track Number
CALL  4014H         ; erase sector
CALL  400BH         ; turn off drive
EI                  ; enable interrupts again
…
```

Sector 12 in track 28 of the disk in drive 2 will be erased.

Internally called routines: WRITE

SVMAP - Save allocation Map to disk.

Call: **CALL 4017H**

Input: The corresponding drive must be switched on.

The current allocation map is in the DOS work area in MAPAREA.

Exit: The sector map was written to sector 15 of track 0 on the floppy disk in the selected drive..

The drive remains powered on.

Registers used: AF, BC. DE, HL

Error handling: In the event of an error, this routine automatically branches to the ERROR routine. Custom error handling is not possible.

The sector occupancy overview (allocation map) is transferred from the corresponding buffer of the DOS work area (MAPAREA) to the data buffer and written from there to sector 15 of track 0 on the diskette.

Note that you must turn the drive on and off by yourself.

Before saving, check that the floppy disk is not write-protected,

Example:

```
…
…
DI                  ; disable interrupts
LD    A,10H         ; select Drive 1
LD    (IY+11),A
CALL  4008H         ; and turn on
LD    BC,50         ; 50 ms delay
CALL  4038H
IN    A,(13H)       ; check write protection
OR    A
LD    A,4
JP    M,400EH       ; read only!
CALL  4017H         ; write back allocation map
CALL  400BH         ; turn off drive
EI                  ; enable interrupts again
…
…
```

The sector allocation overview is written back from the DOS work area (MAPAREA) to the diskette in drive 1 (track 0, sector 15).

Internally called routines: WRITE

INIT - Initialize disk.

Call: **CALL 401AH**

Input: DK (IY+11) = Drive identifier.
X'10' = Drive 1
X'80' = Drive 2

Exit: The floppy disk in the selected drive has been initialized.

Registers used: AF, BC. DE, HL

Error handling: In the event of an error, this routine automatically branches to the ERROR routine. Custom error handling is not possible.

A diskette in the selected drive is reinitialized, i.e. it is divided into 40 tracks with 16 sectors each and provided with the appropriate synchronization and identification marks.

All data previously on this diskette will be erased.

This routine handles the powering on and off of the drive itself.

The write protection is checked by INIT, the interrupts are switched off at the beginning.

Example:

```
…
…
LD     A,10H              ; select Drive
LD     (IY+11),A
CALL 401A                 ; initialize disk
EI                        ; enable interrupts
…
…
```

The floppy disk in drive 1 is initialized.

Internally called routines: IDAM
STPIN
STPOUT
DLY

CSI - Interpret command parameters.

Call: **CALL 401DH**

Input: HL = starting address of a file name enclosed in double quotes.
This must end with X'00' or ':'.

Exit: The file name was checked and transferred to the FNAM field of the DOS vectors.

HL = address of terminator

.

Registers used: AF, B. HL

Error handling: If the file name is not enclosed in quotation marks or not correctly terminated, this routine branches to BASIC and the message "?SYNTAX ERROR" is displayed.

The first eight characters of a filename enclosed in quotation marks are transferred to the FNAM field of the DOS vectors.

A BASIC line end identifier X'00' or a command separator ':' must be located after the closing quotation mark.

This routine is used by DOS-BASIC to check the syntax.

Disk access does not take place.

Example:

```
            …
            …
            LD    HL,DNAM1          ; address of filename
            CALL 401DH              ; are transmitted to FNAM
            …
            …
DNAM1       DEFM      '"MAILBOX":'
```

The file name "MAILBOX" is transferred to the DOS vector FNAM for subsequent addressing.

HEX - Conversion ASCII to HEX.

Call: **CALL 4020H**

Input: HL = Start address of a 4-byte hexadecimal number
in ASCII format..

Exit: DE = equivalent hexadecimal value (binary)
HL = Input address + 4

Registers used: AF, DE, HL

Error handling: Carry = 0 - no error
Carry = 1 - conversion failed

This routine can be used to convert a hexadecimal address input from the keyboard to its binary equivalent.

This routine is used by DOS BASIC, e.g. by the BSAVE command, to interpret and accept the program start address and end address.

Example:

```
        …
        …
        LD    HL,ASCII        ; Address ASCII value
        CALL  4020H           ; convert
        JR    NC,A1           ; Carry=0? ok, go to A1
        LD    A,1             ; Carry=1, "SYNTAX ERROR"
        JP    400EH           ; output via ERROR routine
A1      LD    (BIN),DE        ; save binary value
        …
        …
ASCII   DEFM      'A31C'
BIN     DEFS      2
```

The character string in the "ASCII" field is converted into the hexadecimal value and stored in the "BIN" field.

IDAM - Look for the address mark on the diskette

Call: **CALL 4023H**

Input: The corresponding drive must be switched on.

SCTR (IY+17) = Sector Number
TRCK (IY+18) = Track Number

Exit: If the return jump is error-free, the read/write head is located directly in front of the data mark of the sector being searched for..

Registers used: AF, BC. DE, HL

Error handling: A = 0 - Address mark found
A = 9 - Address mark not found
A = 17 - BREAK key pressed

if A = 0, Z flag is set
if A <> 0, Z flag is clear

.

This routine is used to position the read/write head in front of the data mark of this sector before writing or reading a sector.

IDAM first positions the head over the desired track and then reads address mark after address mark until the desired sector is found. The writing or reading process for the data field must then begin immediately, since the diskette continues to rotate.

IDAM is already integrated in the READ and WRITE routines for reading or writing a sector.

Example:

```
…
…
DI                      ; disable interrupts
LD    A,80H             ; select Drive 2
LD    (IY+11),A
CALL  4008H             ; and turn on
LD    BC,50             ; 50 ms delay
CALL  4038H
LD    (IY+17),6         ; Sector Number
LD    (IY+18),14        ; Track Number
CALL  4023              ; search sector
JP    NZ,400EH          ; error or BREAK
…
…          Read or write sector
…
```

The read/write head should be positioned in front of the data mark of sector 6 of track 14 for subsequent reading or writing.

Internally called routines: STPIN
STPOUT

CREATE - Write an entry in the table of contents

Call: **CALL 4026H**

Input: filename in FNAM (IY+1)
File Type in TYPE (IY+9)

The allocation map must be loaded (MAPAREA).

The drive must be powered on.

Exit: The file was entered in the table of contents, the first data sector for this file was reserved.

Registers used: AF, BC. DE, HL

Error handling:

A = 0 Entry made
A = 2 File already exists
A = 3 no space in the table of contents
A = 7 no free sector (disk full)
A = 9 An address mark was not found
A =10 A checksum error occurred during reading
A =17 BREAK key pressed

An entry is made in the directory for the file specified in FNAM and the first free sector is reserved for this file.

First, SEARCH checks whether a file with the same name already exists. If not, a free entry in the table of contents is determined with FIND and a first free sector is searched for with MAP. If both were successful, the entry is made in the table of contents.

To do this, the file type, separator ':', file name and the address of the first sector are entered in the 16-byte entry found in the table of contents, and the table of contents is written back.

The allocation map should then also be written back to the diskette, otherwise the first sector will not be definitively assigned. If you forget this, it will later lead to a double allocation of this sector and possibly to an inextricable mess of the data.

Of course, rascals can take advantage of this and make two different entries for the same file with different names in the table of contents. Sensible ???

Before calling CREATE, you should definitely check the write protection of the diskette.

The success of the action can be checked by evaluating the A register.

Example:

```
            …
            …
            DI                  ; disable interrupts
            LD    (IY+11),10H   ; select Drive 1
            CALL  4008H         ; and turn on
            LD    BC,50         ; 50 ms delay
            CALL  4038H
            IN    A,(13H)       ; check write protection
            OR    A
            LD    A,4
            JP    M,400EH       ; read only!
            CALL  4011H         ; load allocation map
            LD    HL,DNAM       ; filenames in the field
            CALL  401DH         ; copy to FNAM
            LD    (IY+9),'B'    ; set Type = 'B'
            CALL  4026H         ; enter the file in the table of contents
            OR    A             ; error occured?
            JP    NZ,400EH      ; yes, to the ERROR routine
            CALL  4017H         ; write back allocation map
            CALL  400BH         ; turn off the drive
            EI                  ; allow interrupts again
            …
            …
DNAM        DEFM      '"KARTEI":'
```

An entry is made in the table of contents of the diskette in drive 1 for the binary file "KARTEI" (type = B).

Internally called routines: SEARCH
FIND
MAP
READ
WRITE

MAP - Detect a free sector on the disk

Call: **CALL 4029H**

Input: filename in FNAM (IY+1)
File Type in TYPE (IY+9)

Exit: NSCT (IY+21) = Sector Number
NTRK (IY+22) = Track Number

The sector addressed with NSCT and NTRK was reserved in internal allocation map (MAPAREA)..

Registers used: AF, BC, HL

Error handling: A = 0 sector found
A = 7 no free sector (disk full)

This routine determines a free sector in the internal allocation map in DOS vectors (MAPAREA), which should previously be filled with the current allocation map from track 0, sector 15 of the diskette.

If a free sector is determined, the corresponding bit in the allocation map is set to 1.

Please note, however, that a final allocation has only taken place when the allocation map has been written back to the diskette.

The result (the sector address) is passed in the NSCT and NTRK fields of the DOS vectors. If you want to access the sector, e.g. with WRITE, you must first transfer this address to the SCTR and TRCK fields.

No disk access is performed by the MAP routine.

Example:

```
…
…
DI                  ; disable interrupts
LD    (IY+11),80H   ; select Drive 2
CALL  4008H         ; and turn on
LD    BC,50         ; 50 ms delay
CALL  4011H         ; load allocation map
CALL  4029H         ; determine free sector
OR    A             ; error occured?
JP    NZ,400EH      ; yes, to the ERROR routine
CALL  4017H         ; write back allocation map
CALL  400BH         ; turn off the drive
EI                  ; allow interrupts again
…
…
```

A free sector is determined and allocated on the diskette in drive 2. The sector address is passed in the NSCT and NTRK fields of the DOS vectors.

SEARCH - Find file in table of contents

Call: **CALL 402CH**

Input: Filename in FNAM (IY+1)

The drive must be powered on.

Exit: If file exists, type of file in TYPE+1 (IY+10).
The sector of the table of contents with the entry found is in the data buffer.
Register DE points to the byte after the name.
SCTR and TRCK contain the address of the sector.

Registers used: AF, BC, DE, HL

Error handling:
A = 0 File does not exists
A = 2 File already exists
A = 9 An address mark was not found
A =10 A checksum error occurred during reading
A =13 File does not exists
A =17 BREAK key pressed

The SEARCH routine checks whether a file with the name stored in FNAM already exists in the table of contents of the addressed diskette.

The result of the search is transferred in the A register.

A = 2 means that there is a corresponding entry
The sector of the table of contents is in the data buffer and DE points to the byte after the name of the entry found (= address of the 1st sector of this file).
The SCTR and TRCK fields of the DOS vectors contain the sector address within the table of contents.
The TYPE+1 (IY+10) field contains the type of the found file.
You may have to evaluate this yourself.

A = 0 or A = 13 have the same meaning.
The specified file is not in the table of contents of the diskette.
A=0 - the end of valid entries has been reached.
A=13 - the end of the table of contents has been reached.

All other values of A indicate an error or fact that BREAK key was pressed.

Example:

```
…
…
DI                  ; disable interrupts
LD    (IY+11),10H   ; select Drive 1
CALL  4008H         ; and turn on
LD    BC,50         ; 50 ms delay
CALL  4038H
CALL  4011H         ; load allocation map
```

```
        LD    HL,DNAM          ; filename text
        CALL  401DH            ; copy filename to FNAN
        CALL  4026H            ; find file in table of contents
        OR    A
        JR    Z,A1             ; unavailable!
        CP    0DH
        JR    Z,A1             ; unavailable!
        CP    2                ; error?
        JP    NZ,400EH         ; yes, to the ERROR routine
        IN    A,(13H)          ; check write protection
        OR    A
        LD    A,4
        JP    M,400EH          ; read only, to the ERROR routine
        EX    DE,HL            ; address of entry in HL
        LD    DE,-10           ; HL to the beginning of the entry
        ADD   HL,DE
        LD    (HL),1           ; release entry
        CALL  4023H            ; write back sector of table of contents
        …
        …          release occupied sectors in the allocation map
        …
A1      CALL  4017H            ; write back allocation map
        CALL  400BH            ; turn off drive
        EI                     ; enable interrupts again
        …
        …
DNAM    '"DIARY":'
```

The "DIARY" file, if present, is deleted from the directory of the diskette in drive 1. If not there, the delete routine is skipped.

Note that this example has not been fully coded out. In addition to deleting the entry in the table of contents, you must also release all occupied sectors of this file in the allocation map.

Internally called routines: READ

FIND - Look for a free entry in table of contents

Call: **CALL 402FH**

Input: The drive must be powered on.

Exit: SCTR = Sector Number
TRCK = Track Number

The addressed sector of the table of contents is in the data buffer.
HL points to the beginning of the free entry.

Registers used: AF, BC, DE, HL

Error handling:

A = 0	Ok, free entry determined
A = 3	no free entry available
A = 9	An address mark was not found
A =10	A checksum error occurred during reading
A =13	File does not exists
A =17	BREAK key pressed

A free entry is determined in the table of contents of the activated floppy disk.

The result is transmitted in register A.

If successful (A=0), the corresponding sector of the table of contents is in the data buffer and register pair HL points to the beginning of the free entry.

The SCTR and TRCK fields contain the address of this sector. An entry can now be made there.

Example:

```
…
…
DI                  ; disable interrupts
LD    (IY+11),10H   ; select Drive 1
CALL  4008H         ; and turn on
LD    BC,50         ; 50 ms delay
CALL  4038H
CALL  402FH         ; determine free entry
OR    A             ; successful?
JP    NZ,400EH      ; no, to the ERROR routine
CALL  400BH         ; turn off drive
EI                  ; enable interrupts again
…
…
```

A free entry in the table of contents is determined on the diskette in drive 1.

Internally called routines: READ

WRITE - Write sector to disk

Call: **CALL 4032H**

Input: The drive must be powered on.

The sector to be written must be in the data buffer.

The address of the sector to be written must be in the SCTR and TRCK fields of the DOS vectors.

Exit: The sector in the data buffer was transferred to the diskette.

Registers used: AF, BC, DE, HL, BC', DE', HL'

Error handling: A = 0 Ok, sector written
A = 9 An address mark was not found
A =17 BREAK key pressed

The sensitive data in the data buffer are transferred to the addressed sector of the diskette. The checksum is determined and placed at the end of the sector.

The sector is transferred including the data mark (10 bytes) in front of the data buffer and the checksum, ie a total of 140 bytes are written to the diskette.

[TODO] 140 Bytes??? Not sure it's true

Example:

```
…
…
DI                      ; disable interrupts
LD    (IY+11),10H       ; select Drive 1
CALL  4008H             ; and turn on
LD    BC,50             ; 50 ms delay
CALL  4038H
IN    A,(13H)           ; check write protection
OR    A
LD    A,4
JP    M,400EH           ; read only, to the ERROR routine
LD    (IY+17),10        ; set Sector Number in SCTR
LD    (IY+18),5         ; set Track Number in TRCK
CALL  4023H             ; write sector to disk
OR    A                 ; error occurred?
JP    NZ,400EH          ; yes, to the ERROR routine
CALL  400BH             ; turn off drive
EI                      ; enable interrupts again
…
…
```

The data from the data buffer is written to track 5, sector 10 of the floppy disk in drive 1.

Internally called routines: IDAM

READ - Read sector from disk

Call: **CALL 4035H**

Input: The drive must be powered on.

Address of the sector to be read in SCTR (IY+17) and TRCK (IY+18) of the DOS vectors.
RETRY (IY+19) = number of read attempts

Exit: The read sector is in the data buffer.

Registers used: AF, BC, DE, HL

Error handling:
A = 0 Ok, sector was read
A = 9 An address mark was not found
A =10 Checksum wrong
A =17 BREAK key pressed

The sector addressed with SCTR and TRCK is transferred from the diskette to the data buffer.

The checksum is determined and compared at the end of the sector with the value stored there.

In the RETRY (IY+19) field of the DOS vectors you can specify how many read attempts should be made if the checksum is incorrect (default value = 10).

Example:

```
…
…
DI                  ; disable interrupts
LD    (IY+11),80H   ; select Drive 2
CALL 4008H          ; and turn on
LD    BC,50         ; 50 ms delay
CALL 4038H
LD    (IY+17),14    ; set Sector Number in SCTR
LD    (IY+18),28    ; set Track Number in TRCK
```

```
CALL 4035H          ; read sector
OR   A              ; error occurred?
JP   NZ,400EH       ; yes, to the ERROR routine
CALL 400BH          ; turn off drive
EI                  ; enable interrupts again
…
…
```

Sector 14 data on track 28 is transferred from the disk in drive 2 to the data buffer.

Internally called routines: IDAM

DLY - n milliseconds delay

Call: **CALL 4038H**

Input: BC = number of milliseconds.

Exit: -

Registers used: AF, BC

Error handling: -

This routine causes a delay whose duration in milliseconds is determined by the entry in the register pair BC.

Example:

```
…
…
DI                  ; disable interrupts
LD   BC,1000        ; 1 sec delay
CALL 4038H
EI                  ; enable interrupts again
…
…
```

Causes a delay of one second.

STPIN - Advance read/write head n tracks

Call: **CALL 403BH**

Input: The drive must be powered on.

B = number of tracks

Exit: The read/write head was set in front of the corresponding number of tracks.
DTRCK (IY+20) contains the new current track number.

Registers used: AF, BC

Error handling: -

The read/write head is advanced by the number of tracks contained in B, but up to track 39 at most.

Example:

```
…
…
DI                      ; disable interrupts
LD    (IY+11),10H       ; select Drive 1
CALL 4008H              ; and turn on
LD    BC,50             ; 50 ms delay
CALL 4038H
LD    B,10              ; put head in front of tracks
CALL 400BH              ; turn off drive
EI                      ; enable interrupts again
…
…
```

STPOUT - Reset read/write head n tracks

Call: **CALL 403EH**

Input: The drive must be powered on.

B = number of tracks

Exit: The read/write head was reset the corresponding number of tracks. DTRCK (IY+20) contains the new current track number.

Registers used: AF, BC

Error handling: -

The read/write head is reset by the number of tracks contained in B, but up to track 0 at most.

Example:

```
…
…
DI                  ; disable interrupts
LD    (IY+11),10H   ; select Drive 1
CALL  4008H         ; and turn on
LD    BC,50         ; 50 ms delay
CALL  4038H
LD    B,5           ; reset head 5 tracks
CALL  403EH
CALL  400BH         ; turn off drive
EI                  ; enable interrupts again
…
…
```

LOAD - Loading a program or memory area

Call: **CALL 4041H**

Input: Filename in FNAM (IY+1)
Filoe Type in TYPE (IY+9)

Exit: 78A4/A5 = starting address
78F9/FA = ending address + 1

Registers used: AF, BC, DE, HL

Error handling:

A = 0	Ok
A = 9	An address mark was not found
A =10	A checksum error
A =12	File found but wrong type
A =13	File does not exists
A =17	BREAK key pressed

The program specified in the FNAM field is transferred from the diskette to the memory.

After successful completion, the start address and the end addresst+1 of the memory area are transferred in the BASIC communication area at address 78A4/A5 and at address 78F9/FA.

This routine only works with files that contain the start address in bytes 12 and 13 of the table of contents and the end address + 1 in bytes 14 and 15, ie not with standard data files.

If a program is saved with the BASIC commands SAVE or BSAVE or by machine programs with the routine SAVE, this entry is made automatically.

The LOAD routine automatically turns the drive on and off and also turns off interrupts.

Example:

```
        …
        …
        LD    (IY+11),80H       ; select Drive 2
        LD    HL,DNAM           ; filename to FNAM
        CALL  401DH             ; register
        LD    (IY+9),'B'        ; file type 'B' (binary)
        CALL  4041H             ; load program
        OR    A                 ; error occurred?
        JP    NZ,400EH          ; yes, to the ERROR routine
        EI                      ; enable interrupts again
        …
        …
DNAM    DEFM    '"GRAFDR":'
```

The binary file or machine program "GRAFDR" is transferred from the floppy disk in drive 1 into memory.

Internally called routines: SEARCH
READ

SAVE - Saving a program or memory area to floppy disk

Call: **CALL 4044H**

Input: 78A4/A5 = Start address of the memory area
78F9/FA = End address+1 of the memory area

The drive must be powered on.

FNAM (IY+1) = file/program name
TYPE (IY+9) = type of file

Exit: The addressed memory area was transferred to the diskette and entered in the table of contents under the specified name.

Registers used: AF, BC, DE, HL

Error handling: If errors occur, the SAVE routine branches directly to the ERROR routine. Own error handling is not possible.

A memory area or program is transferred from memory to floppy disk. Start and end addresses of data to be transferred are in the corresponding BASIC pointers 78A4 and 78F9.

This adresses are entered in the table of contents under the name specified in FNAM and the type entered in the first byte of TYPE.

You must first switch on the floppy disk yourself; it is turned off by the SAVE routine when the save process is complete.

You should also check the write protection beforehand.

Example:

```
        …
        LD    HL,8000H          ; start address
        LD    (78A4),HL
        LD    HL,9000H          ; end address + 1
        LD    (78F9),HL
        LD    HL,DNAM           ; filename to FNAM
        CALL  401DH             ; register
        LD    (IY+9),'B'        ; file type 'B' (binary)
        LD    (IY+11),10H       ; select Drive 1
        CALL  4008H             ; and turn on
        IN    A,(13H)           ; check write protection
        OR    A
        LD    A,4
        JP    M,400EH           ; read only, to the ERROR routine
        CALL  4044H             ; write memory area to diskette
        EI                      ; enable interrupts again
        …
DNAM    DEFM    '"TESTDAT":'
```

The Speicherhereich 8000 - 8FFF is transferred to the diskette under the name "TESTDAT" with type "B".

Internally called routines: READ CREATE
MAP SEARCH
WRITE

To supplement these routines, two more functions are listed here that you will find in many of the previous examples.

DRIVE - Selecting a drive

This function cannot be accessed via the jump table.

However, it is easy to do as you just need to put the correct code of the selected drive in the DK (IY+11) field of the DOS vectors.

Drive 1 = LD (IY+11),10H

Drive 2 = LD (IY+11),80H

This code is used by the PWRON routine to select the correct drive and turn it on.

WPROCT - Check write protection

In many cases, you are responsible for checking the write-protection status of a diskette before performing a write operation.

You can get the information about this via port 13H. If the diskette's write-protection notch is taped over, bit 7 of this port is set to 1.

To do this, the drive must be selected and switched on.

Example:

```
…
…
IN      A,(13H)         ; read in port 13
OR      A               ; check byte
LD      A,4             ; set error code
JP      M,400EH         ; if negative, then the disk is
                        ; write-protected.
…
…
```

If the diskette is write-protected, error code 4 branches to the ERROR routine and the message "DISK WRITE PROTECTED" is output there.

Mein Home-Computer

Das Magazin für aktives und kreatives Computern

Monat für Monat über 30 Seiten Programme

Und das bringt

Mein Home-Computer

jeden Monat:

- ✻ Programme für alle gängigen Home-Computer
- ✻ Anwendungsbeispiele aus der Praxis
- ✻ Marktübersicht, Tests und Kaufberatung für Zusatzgeräte und Home-Computer
- ✻ Schnellkurse für Einsteiger zum Sammeln
- ✻ Tips und Tricks
- ✻ Interessantes, Aktuelles und Unterhaltsames aus der Home-Computer-Szene
- ✻ News, Clubnachrichten

Holen Sie sich die neueste Ausgabe bei Ihrem Zeitschriftenhändler oder fordern Sie ein Kennenlernheft direkt beim Vogel-Verlag, Leserservice HC, Postfach 67 40, 8700 Würzburg, an.

Die Einsteiger-Modelle für Schüler und Studenten

LASER™
HOME-COMPUTER

LASER 210, 8 KByte RAM,
erweiterbar um 16 oder 64 KByte,
8 Farben, Programmsprache BASIC.

LASER 310 mit gleicher Ausstattung wie Laser 210,
aber 18 KByte RAM und mit Schreibmaschinen-Tastatur.

Floppy Disk Controller für 2 Laufwerke
mit LASER-DOS, Speicherkapazität 80 KByte.

Generalimporteur: CE ○ TEC Trading GmbH
Lange Reihe 29, 2000 Hamburg 1

aktiv und kreativ computern

Früher oder später wird bei jedem Computerbesitzer der Wunsch nach einem Diskettenspeicher übermächtig. Zu groß sind die Vorteile des direkten Zugriffs gegenüber der sequentiellen Arbeitsweise. Nur so kann man alle Möglichkeiten seines Geräts voll ausschöpfen.

In diesem Band wird das Disketten-Betriebssystem des Laser-Computers in seinem Aufbau und seiner Anwendung detailliert beschrieben. Neben den BASIC-DOS-Befehlen werden auch die Schnittstelle und Anwendbarkeit in Maschinenprogrammen erläutert. Anwendungsbeispiele erleichtern den Einstieg in die Diskettenwelt.

VOGEL-BUCHVERLAG
WÜRZBURG

ISBN 3-8023-0868-9