Business Computer

MZ-3500

EOS 3.0®
MANUAL

SHARP

E O S

Operating System for
Personal-Computers
with a Z8Ø-CPU

Version 3

User's Manual

Section A

General Introduction

- Contents -

Welcome to the Family of Computer Users!

So you have been selected to make yourself acquainted with the operation of your new computer system. Before you work through the following pages, you may want to study the operation instructions of the computer system. When you know which plug belongs into which jack, where to switch on the various devices and what to observe to prevent damage to the new apparatus, we may continue.

Right, here are some hundred pages of our User's Manual ahead of you. You will have to learn much of this material in order to operate your new gear off hands.

Do not be afraid, you will manage it, certainly! The very best method to get accustomed to your new environment is by hands-on experience. Just put this manual next to your CRT screen and little by little, try everything. Beforehand, you may want to read through this book to get an overwiew.

```
*******************************
*     General Introduction    *
*******************************
```

A.1. What is EOS like?

The term EOS designates the operating system of your computer. It consists of a package of programs which enable you to do meaningful things with the computer. Without such kind of programs, a computer were hardly more than a useless conglomeration of some electronic and mechanical devices, at best suited to enlight a tinkerer's minds.

The operating system programs perform highly fundamental tasks. Some of them are always active without you being aware of it. Whenever you press a key at your keyboard, when a character is output on the screen, even when you switch on the machine and think that nothing at all happens, it is always the operating system busy at work.

A number of the programs have names of their own. Later on, you will get acquainted with their names and functions. Some other are buried deep into the system's interior and only their programmer knows their names and what they are good for.

To cut a long story short: All those programs not directly related to the problem you are working at, but indispensable to keep the computer running, are operating system programs.

EOS is an outstanding operating system in many respects. It tries to give all advice and messages in plain English, German, French or Italian. It has built-in specific functions to assist a technician in troubleshooting, should the system once fail. It works faster, more thoroughly and more safely than other comparable operating systems.

EOS comes along with utility programs to enhance its operating comfort, data security and practical usefulness.

EOS is capable of operating your printer and at the same time, run other programs. The era of waiting for the printer to finish has come to an end, at last! You may do text processing, inventory management or accounting, while the printer is busy listing the results of other program runs.

```
*******************************
*     General Introduction    *
*******************************
```

A.2. EOS and its Relatives

Why keep it secret: EOS has been created with the facilities of another operating system in mind - the CP/M system by Digital Research Corp., California, USA.

There is some very good reason to do so. It is the CP/M system which has established a worldwide accepted standard for programs. There are several thousand programs available, all of them being executable on every computer with CP/M or a compatible operating system installed on it.

In some way, you may compare the standard to the standardized connection of a turntable to your HiFi set. You only have to imagine that this "plug" has much more connectors.

Using a computer system with EOS installed on it makes you become a member of a worldwide community of computer users, their number currently approaching the 1,000,000 figure. Day by day, this number keeps on growing, and so does the number of good application programs which can be run on all these computers.

```
********************************
*     Outside the Computer     *
********************************
```

A.3. Outside the Computer

It is difficult to say anything about outer appearance and function of particular devices, because one cannot predict the ideas which will come to the minds of designers of new computer systems. We will try our best to describe the functions of those devices which are generally found with a computer running under the control of EOS. Indeed, some of these devices are strictly indispensable to make EOS work.

A.3.1. The CRT Screen

You cannot renounce a CRT screen. Without a CRT, it would be impossible to communicate with the computer. It displays in readable form the characters the computer (or better, the program running currently in it) sends to it. Because the information is transferred character by character, the CRT is classified as a character oriented device. We will encounter this term with some other devices, too.

From the operating system's view, the CRT is denoted by the symbolic name CONOUT:, which is an abbreviation of CONsole OUTput. Because the console, being the classical input/output device, transfers the operators input to the computer, the CRT cannot do without its all-time companion:

A.3.2. The Keyboard

Like the CRT, the keyboard is absolutely necessary for your computer and the EOS operating system running on it. Due to its strong relation to the CRT, it is linked like the CRT by a cable to the main-unit of your MZ-35ØØ.

With the keyboard, you can enforce your will on the computer. Here you enter your commands. This is also done character by character, with each key you touch with your fingers. Hence, the keyboard is a character oriented device, too.

The keyboard is known by the operating system by its symbolic name CONIN: (from CONsole INput) and represents the input channel of the console.

****
* Outside the Computer *
****

A.3.3. The Printer

The printer is a representant of those devices which only can receive data from the computer. Though you can use the computer without a printer - there are numerous programs which do not require a printer, but do very useful things - a printer is indispensable for most serious applications.

The printer receives the text to be printed character by character, too. Its symbolic name reads LSTOUT:, from LiST OUTput.

A.3.4. The Auxiliary Device

Besides the aforementioned devices, EOS provides for a free definable auxiliary device. This can be used for data communications to another computer via a telephone link, for example. The auxiliary device is divided into two channels:

AUXIN: - the input channel and
AUXOUT: - the output channel

You will meet these names again when using the => $DEVICE utility. As a user, you will seldom or never be bothered by these terms.

A.3.5. Mass Storage Units

This is a collective name for all those units which let you permanently store data, programs, texts, etc. This kind of storing is one, reading the data stored another function of these units.

A computer system must be equipped with at least one storage unit, if EOS shall be run on it.

Physically, the storage units are Winchester or disk drives installed in your computer, or connected to it with a cable. EOS can manage up to sixteen such units.

It is done better not to speak of devices when referring to storage units. Usually, a large harddisk is divided logically into several storage units, which are optically in one case. Disk drives are generally mounted in pairs in the computer's, or in a seperate, cabinet. Hence, we will speak of "drives" from now on.

****
* Outside the Computer *
****

Drives have no specific names, but are denoted by a letter, followed by a colon. Thus, the first drive is named A: , the second B: , up to drive P: .

Depending on the configuration, a maximum of eleven drives may be defined on your MZ-3500. Generally, the fastest drive comes first. If you are happy owner of a Winchester disk, this is referred to as both drives A: and B:. Standard floppy disks come next to the harddisk, mini floppy drives are the last ones. The built-in RAM disk is firmly installed as drive P:.

As an example, this is the configuration of a computer with two mini floppy drives:

A: Right disk drive
B: Left disk drive
P: RAM disk

A system with a standard dual floppy drive and two mini floppy drives would look like:

A: Standard floppy, right drive
B: Standard floppy, left drive
C: Mini floppy, right drive
D: Mini floppy, left drive
P: RAM disk

A system with a Winchester disk, a standard floppy dual drive and two mini floppy drives would have the following configuration:

A: Winchester disk
B: Winchester disk
C: Standard floppy, right drive
D: Standard floppy, left drive
E: Mini floppy, right drive
F: Mini floppy, left drive
P: RAM disk

Winchester and Floppy disk drives have one feature in common: data are stored on them in entire blocks. With each access to these storage units, a full block of data is written resp. read. As a consequence they are classified as block oriented devices.

****
* Outside the Computer *
****

A.3.6. Handling Diskettes

Concerning diskettes, you should observe the following: always be very careful with the main switch. It can happen that the computer runs out of control for a very short time when switched off, writing some nonsense on the disks. This nonsense is inevitably lethal to data and programs stored there. Besides, the electronics and mechanics can be damaged by switching off power during reading or writing. We should better postpone such difficulties for cases beyond our responsibility.

> Always remove the diskettes from the drives before switching off power! Never switch off as long as an acitivity light at a disk or Winchester drive is on!

Diskettes are highly sensitive. You should not try how sensitive they are by handling them carelessly.

> Never touch the exposed areas of the magnetic surface with your fingers!
>
> Do not use a hard pencil to write on the disk labels. It is good practice to inscribe the labels before attaching them to the disk jackets!
>
> Always store diskettes in a protective envelope!
>
> Be careful with the magnetic influences of transformers, motors, loudspeakers, TV sets, telephones, etc!
>
> Do not expose them to direct sunlight or other sources of heat, keep them cool!
>
> Do not use damaged diskettes any longer, dispose them!

There are good diskettes, and there are bad ones. Inexpensive diskettes need not be bad, whereas very good diskettes cannot be cheap. you should decide to use nothing else but very good diskettes. You will know why when a disk error spoils the work of several days, or even weeks! Besides, the very good ones cause less wear to the read/write heads of your drives. On the long term, they are the least expensive data media!

```
*******************************
*     Outside the Computer    *
*******************************
```

A.3.7 The Central Unit

This term refers to the mysterious part of your computer where everything comes together. Quite literally so, because it is here where the cables from the CRT monitor, the keyboard, the printer and the mass memories end. Your MZ-3500 keeps up to two Minifloppy drives in this housing, too.

The central unit consists of the following functional blocks:

- The CPU

  In order to run EOS, this must be a chip from the Z80 family (Your MZ-3500 includes two of such chips for better throughput). It transforms the instructions given in the programs into the desired actions and manages the main memory. That is all you need to remember for now.

- The Main Memory

  All you should remember is that the programs are executed from within the main memory. EOS also resides here. Its capabilities are related directly to its size. As the Z80 can address only 64 Kbytes directly, EOS has several memory management mechanisms built in, thus being able to make full use of the 128 to 256 Kbytes of your MZ-3500.

  Data and programs are stored in the main memory. Each single byte can be directly referred to by the CPU, can be read, changed, and stored back again. Its disadvantage is its energy dependance; as soon as power is turned off, its contents get lost at once.

- The Control Units

  These devices, some of them being very "intelligent", are devoted to control data transfer to and from external devices.

```
*******************************
*     Inside the Computer     *
*******************************
```

A.4. Inside the Computer

Up to here, we have been talking about real-world matters, things which can be touched by the hands. We will now devote ourselves to some things which are inseparably connected to data processing but not as easy to explain, because one cannot see them. Very interested readers should refer to detailed special literature, as we can only give a brief overview at this place.

A.4.1. The Bit

Computers are dumb but swift. A knowledgeable man once called them "fast idiots". A computer knows only two logical states. It cannot even count to two, if you want to see it this way, it is limited to one. It has thus just two possibilities:

```
  Ø or 1
 ON or OFF
YES or NO
```

this is here the question (with apologies to W.S.). These two states can be represented by a bit. A bit is the smallest quantitiy the computer can act on. Because a bit was somewhat too tiny for human use, man concatenated an arbitrary few of them and called this new quantity

A.4.2 The Byte

Just eight bits form a byte. A byte is the smallest quantity the computer can refer to in its memory. Not disregarding zero, these are exactly 256 different states - one can count from Ø to 255 with 8 bits.

That does not seem to be much, at least at the first glance. But let us consider what there can be put into the figure 256. We will not at all exhaust the range when representing the alphabet, because this has far less than 256 characters, even if we add digits and special characters.

We simply remember that a character occupies just one byte, that will anytime be correct.

To provide an elegant means to handle larger amounts of characters, new terms have been invented:

```
KB = Kilobyte    =    1Ø24 bytes
MB = Megabyte    =    1Ø24 KB   =    1,Ø48,576 bytes
```

```
*******************************
*     Inside the Computer     *
*******************************
```

A.4.3. The Field

This term relates more to application programs than to an operating system. We shall explain it here for completeness.

If several characters belong together logically, they form a set which may be given a name. Let us assume you want to store your customers' addresses. An address consists of the following data:

Name
Prename
Street
Location

Here, the field "name" may assume any contents. Whether the name reads Smith, Walker, or O'Brian, this is determined by you when you type in the names.

A.4.4. The Record

A record is a logically connected set of fields. To continue the example above, the fields name, prename, street, and location together form the record "address". Whenever your program stores such a record, its contents are determined by the sequence of fields it consists of.

Characters form fields, these form records, and the latter ones form

A.4.5. Files

Simple, isn't it? Formerly, all addresses would have been written on cards and stored in a card-index box, called "customers' index". Thanks to the technical progress, we would nowadays speak of a "customers' file".

Because we got on so quick, lets us have a small break at this moment.

Programs operate on files. They read files, write them, change, rename and erase them. There are programs which perform all of these operations on several files at the same time. An bookkeeping system is certainly an example for this kind of programs.

```
*******************************
*     Inside the Computer     *
*******************************
```

The operating system provides the necessary means for such programs. We have mentioned the international standard EOS is in accord with. Part of this standard is the notation of filenames - more about that later on.

One fact to remember: Programs are files, too. Files containing instructions for the CPU, as opposed to your address data. A specific function of your operating system loads these program files to a precisely determined location of the main memory and advises the CPU to execute the instructions deposited there.

```
*******************************
*          Cold Start         *
*******************************
```

A.5. Operating the Computer

It's becoming serious now! At least now you should read the owner's manual of your new computer or have the vendor demonstrate how to put the machine into operation. We assume that everything is wired accordingly, so please look for the mains switch, as we are going to do the first step:

A.5.1. The Cold Start

Open the disk drive doors and switch the computer on. You will read the message "SYSTEM LOADING" on the screen and a little later, "NO SYSTEM MEDIA". Now insert your EOS system diskette into the slot of the right mini disk drive and close the door.

Did anything happen? No - then it was not the correct drive, or the disk has been inserted upside down - or it just was not the system disk. Try to vary a bit. When after all, you did not succeed, you may want to refer to the owner's manual again. Four of five troubles at this moment are caused by insufficient information.

If everything went right, the message

```
EOS Version 3
Copyright (C) 1983 Daeumling & Zimmermann

Module    Bnk   Load   Strt   Size
SUBCPU     1    4000   4000   25FD
SYSVAR     0    FE00   FE00   0200
SYSTEM     1    2000   3788   1FFA
MZ3500     2    2000   2000   160B
SHELL      0    8000   0100   2062
BANKMGR    0    F100   F500   0500

>
```

should appear on the screen.

```
*******************************
*          Cold Start         *
*******************************
```

These lines are the so-called EOS boot protocol. The operating system consists of a set of files all of which are loaded in sequence into the various memory banks of your computer. The protocol is more of optical value. To the knowledgeable, it indicates the memory bank, the loading address, the start address, and the size of each module loaded. It is more essential that those six modules are loaded from an equal number of files. The names of these files are:

```
SUBCPU.SYS      SYSVAR.SYS      SYSTEM.SYS
MZ3500.SYS      SHELL.SYS       BANKMGR.SYS
```

All these files must be present on the system disk so that EOS can be loaded correctly. If but one of them should be missing, EOS cannot do its job. In order to never lose a file, you ought to create a backup copy of your system disk by means of the => COPYDSK utility program.

The ">" character is, contrary to the message mentioned above, not typical for a cold start but the "ready" prompt of your EOS operating system. It signalizes that EOS is ready to accept your instructions. Essentially, the prompt is not displayed by EOS itself but by the command interpreter SHELL. More on that later on.

What in the world happens during the "cold start"?

Your computer is pre-programmed for this function. Contrary to the programs which can reside on one of the drives, the "cold start loader" (IPL) or "bootstrap" program is literally burnt firmly into the memory of the central unit and can never get lost. It is very short, doing not much more than read the various parts of the EOS operating system from drive A: and deposit them in memory. It then passes control to EOS, which resets the peripheral controllers to their initial state, displays the nice message and loads the system part (namely the SHELL program) which displays the ready prompt and waits for your orders.

Some people still wonder what the term "bootstrap" means, at least in connexion with computers. As many other terms in data processing, it orginated from over the big ocean, where people prefer a pregnant and sometimes drastic language. Do you remember how to put on boots? Yes, there are two small straps at the shafts you seize and pull at, and something greater (the boots themselves) follows. And the computer, when switched on, "seizes" the tiny "bootstrap" loader, which "pulls" somewhat greater (the operating system) in.

```
*******************************
*          Cold Start         *
*******************************
```

A.5.1.1. The Auto Start Command

In this chapter, we are going to discuss a specific EOS facility which properly requires some knowledge of commands and filenames to be understood. But as this matter occurs only at a cold start of EOS, it must be described here. You may want to read this chapter thoroughly after you know more on EOS.

EOS allows for a number of commands to be executed automatically immediately after it has started itself. Such commands are entered in a file called "PROFILE.SUB" on drive A:. If this file is present, the command

```
$DO A:PROFILE
```

is automatically executed after EOS has started. For details on the structure of a ".SUB" file, please refer to the description of the $DO utility. Just to give a short example: You want all menu files (what menu files are like, can be found in section B) to be copied automatically to the RAM disk P:. Therefore, you create a "A:PROFILE.SUB" file with the following contents:

```
$COPY *.MNU P:
```

This causes the program $COPY to be loaded and executed automatically.

```
********************************
*          Warm Start          *
********************************
```

A.5.2. The Warm Start

The computer is up and running. We see the ready prompt on the screen and could start working. This is correct, but for completion, let us put a few words on the operating system warm start right here.

As already mentioned, the ready prompt is the sign of life of a specific operating system part. This part of EOS is called "SHELL", because it represents in some way the outer "shell" of the operating system. One part of an operating system is embedded into the other, just like the shells of an onion. The task of SHELL is to be at your service and convert your input into meaningful actions.

After a program has been loaded, the presence of SHELL is no longer necessary, it is even undesirable in order to free up memory space for other programs. Hence, it is usual that programs simply ignore SHELL and overwrite it. After they have finished, they call upon the part of EOS which has remained in memory (and cannot be renounced) to re-load SHELL from drive P: (the RAM disk) and pass control to it. Therefore, the usual SHELL prompt re-appears on the screen after each ordinary program termination.

This kind of re-loading is called "warm start" or "warm boot". As SHELL is a relatively small program, this is done within a moment. Should you insist on having your system crash, just erase the file "SHELL.SYS" on drive P:. However, the file is protected against such foolish things by having the file attributes SYSTEM and R/O set. For details on file attributes, please refer to the chapter on filenames.

```
*******************************
*      Changing Diskettes     *
*******************************
```

A.5.3. Changing Diskettes

As an unprepared computer user you certainly think that this is simplicity by itself. Right, one just opens the drive door, one grasp, and there we are. Princially, yes, but....

When you are running an application program for word processing or accounting, you should change a diskette only after the program inquires you to do so.

But why all that?

For reasons of data security, EOS checks with each access to a drive if the same data medium is still present there. If it detects that the diskette has been changed without having been notified so, it refuses to write on this disk and makes known of its indignation by a system error message.

Due to the large number of programs which may be run under EOS, this mechanism is an absolute necessity to maintain data integrity. But when you change diskettes as long as EOS waits for a command, nothing bad can happen. As a new command is to come, EOS assumes that new data media are required for this new job. So there is no reason to be afraid of disk changes before you enter an EOS command!

****
* Using the Keyboard *
****

A.5.4. Using the Keyboard

Next to the system ready prompt ">" there is a small block lit up on the CRT screen. This block is of utmost importance. It signalizes the place on the screen where the next character you type will be displayed. Just try: Whenever you type a character, this appears right at the place of the block, and the block moves one position ahead. The tiny block, which may be blinking sometimes, is called the "current cathode ray tube screen output position indicator". As hardly any human being gifted just a bit less than a genius can memorize this monster of a word, the term has been abbreviated as "cursor". You should remember it well, as you are to encounter it everywhere throughout this manual.

Whenever you press a letter key, the letter will appear on the CRT screen. You can transmit your orders to EOS as if you were operating a typewriter. You type a message and send it by pressing the ENTER key. Henceforth, we will use the character sequence <ENTER> when referring to the ENTER key. Whenever you see this term in the text, this means that you are expected to press the ENTER key at the place specified.

Besides the normal (and partially somewhat extraordinary) characters there is a whole number of characters which cannot be displayed or printed, namely the control characters which control screen display and printer operation. The "carriage return" character is one of them. It is sometimes necessary to enumerate these characters according to the alphabet. Additionally, there is a key labelled "CTRL" near the left edge of the keyboard. If you press the <CTRL> key along with a letter key - regardless whether uppercase or lowercase mode is selected, a control character is generated.

As an example, let us generate the control character BEL (Bell). In general computer language, the BEL character is known as Control-G. Just press the Control key along with the "G" key. You will hear a beep.

To denote CONTROL characters, we will use the up-arrow from now on. In other documentations, you may find the abbreviation CTRL as well:

^G or CTRL-G means: Type CONTROL-G at this place.

```
*******************************
*      Using the Keyboard     *
*******************************
```

A.5.5. BREAK Characters

Typing one of these characters cause the current program to be aborted. They must be entered as the first character of an input line. The standard BREAK character is ^C (Control-C). Furthermore, ^[or <ESCAPE> is another commonly used BREAK character. By means of => $CONFIG, you can define any second BREAK character you prefer. Whenever EOS recognizes a BREAK character, the following message is displayed on the screen:

```
<Restart>
```

To user progams, EOS offers a facility to suppress the recognition of BREAK characters; so do not be surprised when under certain circumstances, typing a BREAK character has no effect! Most EOS utilities, for example, ignore BREAK characters or admit them only at certain situations.

A.5.6. Suspending Screen Output

Many programs do their CRT screen output at such a tremendous speed that no normally gifted man can follow up in reading. Therefore, it was necessary to define a specific character to stop screen output. This is the

^S (Control-S).

character. When you press ^S, screen output is immediately suspended. After output has stopped, you can choose from several possibilities:

^Q (Control-Q) resumes screen output.

^C (Control-C) or another BREAK character aborts the program.

^P (Control-P) switches the printer on. It then runs parallel to the screen until it is switched off by another ^P, or by the SHELL command LIST OFF.

Here too, it shall not remain a secret that a number of programs simply ingnore Control-S. The EOS utility => $TYPE is an example of this class. It cannot be suspended by typing Control-S, for the very simple reason that it stops by itself after it has filled a screen page.

```
******************************
*         Filenames          *
******************************
```

A.6. Filenames

We already mentioned in the general introduction that the notation of filenames is a part of the worldwide standard of EOS and other CP/M compatible operating systems. As practically all commands refer to files in one way or the other, the time has come to talk about filenames.

A.6.1. Unambiguous Filenames

A file designation is composed of two parts, the proper filename and the file type:

```
CUSTOMRS.DTA        ACCOUNTS.REC        D12345
```

These are some typical examples of file designations. Left of the period, the filename is specified, and right of it, the file type. The filename may be up to 8 characters long, the file type up to three. In some cases, the file type may be omitted. We shall deal with them later.

With a few exceptions, all printable characters may be used for constructing filenames. Control characters are not permitted, neither are the characters listed below:

```
= (Equals sign)
< (Less-than sign)
> (Greater-than sign)
: (Colon)
. (Period)
, (Comma)
Space ( blank )
[
]
```

because they have a special meaning with commands.

Furthermore, a number of programs require filenames beginning with a letter, followed by nothing else but letters and digits. Certain file types can be prescribed. For details on these matters, please refer to the corresponding program descriptions.

In the EOS command level, all lowercase letters used in input are converted to uppercase letters. You cannot directly refer to a file which contains lowercase letters in its name (There are some programs using lowercase letters in filenames).

```
*****************************
*         Filenames         *
*****************************
```

Here are some examples for valid filenames:

ADDRESS1.MST FILE1 FILE2.BAK

whereas the following filenames are erroneous:

ADDRESSES.MST = the filename is longer than 8 characters. The remainder is ignored.

FILEX:1.BAK = contains a colon, which is not permitted with filenames.

INVENT. Ø1 = contains a space character in the file type, which is not permitted at that place.

A.6.1.1. File Types

Let us discuss file types a bit further, because in many cases, it has a specific meaning.

You can form a multitude of combinations out of the three characters available with the file type. Let us begin with some file types which are important with EOS and its international standard, and which should never be used for another purpose than intended by the inventor:

.COM = denotes a file which contains an executable program. Such programs can directly be started from the EOS command level. In section C of this manual we will introduce the utility programs to you which belong to the EOS scope of delivery.

.SUB = Files of this type contain command sequences, which can be converted to commands by means of the $DO utility.

.SYS = these files contain parts of your EOS operating system.

.RSX = EOS function module (cf. system manual, section A).

.BAK = denotes a backup file, which contains an old version of a dataset. Word processors automatically generate such files. A special function of the => $COPY utility accounts for this feature. You should always use this file type as intended by its creators.

```
*****************************
*         Filenames         *
*****************************
```

.$$$ = another file type denoting temporary auxiliary files. This type, too, has a special meaning with => $COPY. All EOS utilities which work with temporary files create files of this type.

.MNU = Auxiliary files for the EOS menu program.

.KEY = Files containing function key definitions.

The following file types have no special meanings with EOS, but they are used uniformly throughout the data processing world. It is good customs to stay in accord with these conventions:

.DTA = This file contains data. The filename should further specify the kind of data stored.

.NDX = This file contains indices to datasets. It is good customs to use the same filename for index and data files to make clear the connexion at the first glance.

.OVR
.OVL = These files contain program segments or overlays, which are loaded by another program during execution. It is usual to denote the relation between the main program and the overlays by accordingly chosen filenames.

.REL
.CRL
.IRL = denotes an object module in a specific relocatable format.
.ERL ASSEMBLERS and COMPILERS create such files, which, in the next processing step (linkage), are then linked to form a .COM file, i. e. an executable program.

.TXT
.DOC = such files contain explanatory texts and other documentations on program fuctions or disk contents.

Furthermore it is usual to specify the source language of program files in the file type. Some compilers and interpreters require specific file types for input as well as for output files:

```
*****************************
*         Filenames         *
*****************************
```

```
.C   = C (the language parts of EOS have been written in)
.BAS = BASIC                .COB = COBOL
.PLI = PL/I                 .PAS = PASCAL
.ASM = ASSEMBLER            .MAC = ASSEMBLER
.ALG = ALGOL                .FOR = FORTRAN
.MMP = MUMPS                .LSP = LISP
```

A.6.2. Ambiguous Filenames

It is becoming a little complex now. When you enter commands, you may also make use of ambiguous filenames. This technique is very useful - you will soon recognize it - and allows to refer to a whole group of files at once.

This kind of ambiguity, however, can be used only with system commands, utility programs and programs specially adapted to ambiguous filenames. It is achieved by specifying "blind characters" or "wildcards" within the filename. The wildcards are:

? - the question mark and
* - the asterisk

We recommend that you make yourself acquainted with these features by means of the => $DIR command.

A.6.2.1. The Question Mark

The question mark represents any single character within a filename or file type.

Let us assume that following files are to be found on the current disk:

```
CUSTMR1.DTA     CUSTMR2.DTA     CUSTMR3.DTA
CUSTMR1.BAK     CUSTMR2.BAK
```

Then, by specifying the filename "CUSTMR?.DTA", you can refer to the files CUSTMR1.DTA, CUSTMR2.DTA, and CUSTMR3.DTA. On the other hand, by specifying "CUSTMR1.???", you would have access to the files "CUSTMR1.DTA" and "CUSTMR1.BAK".

****
* Filenames *
****

A.6.2.2. The Asterisk

The asterisk represents a sequence of one or more characters and makes typing of several question marks superfluous.

.

denotes all files on the current drive. Let us stay with the previous example and the files as before:

CUSTMR1.DTA CUSTMR2.DTA CUSTMR3.DTA
CUSTMR1.BAK CUSTMR2.BAK

Then, if we specify the file designation

*.DTA we would refer to the files

CUSTMR1.DTA CUSTMR2.DTA CUSTMR3.DTA

By specifying

CUSTMR1.* we are able enclose the files

CUSTMR1.DTA CUSTMR1.BAK

in our command without having to enter eight question marks. But it is rewarding to spare just two qestion marks, as the following example for using "*" in filename and file type demonstrates:

K*.D* results in

CUSTMR1.DTA CUSTMR2.DTA CUSTMR3.DTA

A.6.3. The Drive

In the previous examples, we have been assuming that the files concerned are found on the current drive, i.e. the drive which EOS indicates in its status line. You shall soon learn how to change the current drive. Here we are dealing with files which reside on another than the current drive.

****
* Filenames *
****

The drive is denoted by one of the letters from A to P, followed by a colon. You remember? - according to your system's configuration, the letters A to P for the 16 drives possible can be used. The drive designation can be specified with filenames. If done so, it must placed at the beginning of the file designation and must be separated from the filename by a colon, so that EOS would not misunderstand it as part of the filename:

A:CUSTMR1.DTA B:CUSTMR1.BAK C:*.BAK A:CUSTMR1.*

The examples all demonstrate the correct usage of drive designations within filenames. It becomes clear now why no colons are allowed in the filename itself.

A.6.4. File Attributes

One or more file attributes can be affixed to each file. Besides name and type, attributes further characterize a file. Normally, file attributes are "invisible", however, an option of the => $DIR command will display file attributes, too. The => $SET utility has been created to set and reset file attributes.

Most attributes are reserved for EOS, however, three of them can be set and reset by you at will:

SYSTEM - This attribute makes a file globally accessible. It can then be read from within other user areas, too. Furthermore, a file with a SYSTEM attribute is not displayed with the directory of a disk, thus improving the readabilty of the list (in some cases, substantially so).

BACKUP - The => $COPY utility sets this attribute of each file which has been secured. On the other hand, EOS resets the attribute automatically if a file has been altered. Thus, files which have not yet been secured can easily be recognized. A special mode of $COPY only copies files without this attribute.

R/O - Files with an R/O (Read Only) attribute can neither be altered, nor can they be erased or renamed.

```
******************************
*         User Areas         *
******************************
```

A.7. User Areas

Each drive can be subdivided into up to 16 so-called user areas, which are numbered from Ø to 15. Normally, you are in user area Ø. But if you feel that there are more files on your Winchester disk than manageable, you can change the user area by means of the command => USER. Programs can only access files within the same user area, with the exception of user area Ø. SYSTEM files (i.e. files with a set SYSTEM attribute) allocated to user area Ø are "global" or "public". They can be erad from any other user area, but they cannot be altered. Therefore you can avoid to keep a copy of each program you need in each user area.

However, if you are using the menu program instead of the SHELL command interpreter, you have bad luck: the menu program does not allow for changing user areas. In this case, you have to stay in user area Ø for all of your activities.

```
*****************************
*      The Status Line      *
*****************************
```

A.8. The MZ-3500 Status Line

The bottom line of your screen has a special function, as already indicated by inverse video mode: it is an information line, not affected by normal screen functions. Either a system status line, or a free definable user status line can be displayed here. Both of them are completely independent. You may switch anytime from one status line to the other and back again.

Let us have a look on what EOS is telling us in the system status line. As you see, the line is divided into eight fields. Let us number the fields from 1 to 8, where 1 denotes the leftmost field, and 8 the rightmost one. Then, the fields and their meanings are:

Field '1' - Name of current program. If the SHELL program is active and you can enter commands, "E O S" appears here.

Field '2' - Current drive.

Field '3' - Active user area. When you are running programs which are switching between drives and user areas, you can follow it up in the status line.

Field '4' - Current keyboard and screen state:

```
' KeyLock '  -  Keyboard is locked, no input possible
' Monitor '  -  Screen in monitor mode
' Key Def '  -  Input of key definitions
' UsrStat '  -  Setting user status line
' SysStat '  -  Setting system status line
'Graphics '  -  Screen in pseudo graphics mode
'Edit Mode'  -  Screen editor active
'Hard Copy'  -  Screen hardcopy is just performed
```

For further information, please refer to section D of the system's manual. The screen in described in detail there.

Field '5' - When EOS executes a command file, the filename appears here.

Field '6' - free for user programs.

Field '7' - Today's date (can be switched off)

Field '8' - Current time (can be switched off)

****
* The Keyboard *
****

A.9. The MZ-3500 Keyboard

Besides the normal keyboard, your MZ-3500 has a number of special function keys. The function keys are free programmable, most of them being predefined by EOS. The numerical keypad is of specific importance in connexion with the CONTROL key. If you press a key of the decimal keypad while holding the CONTROL key down, you can release various functions. But remember: these functions are available only with the numerical keypad! The number keys of the normal keypad have no effect at all in connexion with the CONTROL key!

The table below explains what functions are available:

CTRL-0 : Your MZ-3500 has a keyboard input buffer of 140 characters length which enables you to type before the computer signalizes that it is ready to accept input. This facility is called "type-ahead". If you feel that you may have typed nonsense, press CTRL-0, and the type-ahead buffer is cleared completely.

CTRL-1 : The effect of the keys <SHIFT> and <LOCK> is interchanged. If you have configured your computer to lowercase mode by means of the => $DEVICE utility, you can obtain a normal uppercase mode by aid of this function (the LOCK key converts the entire keyboard to uppercase mode, including the special characters).

CTRL-2 : Select the extra characterset (in some countries available as an option).

CTRL-3 : Switch clock display in status line on / off.

CTRL-4 : Perform a hardcopy of the screen contents onto a printer via the parallel interface.

CTRL-5 : Graphics hardcopy on the SHARP ink jet printer IO-700.

CTRL-6 : Switch high-resolution graphics display off.

CTRL-7 : Start of manual key definition.

CTRL-8 : Terminate manual key definition.

CTRL-9 : Re-initialization of screen and keyboard. The standard key definitions are re-activated.

****
* The Keyboard *
****

A.9.1. Definition of Function Keys

By means of the keyboard functions CTRL-7 and CTRL-8, you can create key functions of your own choice. Just follow the explanations:

1. Press CTRL-7. A greater-than sign ">" will appear on the screen.

2. Press the key you want to redefine. The key is indicated in plain text. For example, if you chose the F1 function key, it reads "F1=". In the fourth field of the status line, the text "Key Def" appears. If the key you selected cannot be redefined, you will hear a beep.

3. Enter the text of your choice. Control characters are indicated by tilt letters.

4. Terminate your defintion by pressing CTRL-8. A less-than sign "<" appears.

If you should do a mistype, you may abort anytime by typing CTRL-7.

Let us have an example. We want to assign the text "Insert!" to the "INS" key near the upper right egde of the keyboard. The procedure looks as follows:

You type	the computer replies
CTRL-7	>
INS	INS=
Insert!	
CTRL-8	<

Whenever you press the INS key, the text "Insert!" will be displayed.

If you want to re-activate the EOS standard key definition, press the CMD key instead of typing text. Up to 128 characters can be associated to any key. However, 300 characters are avilable for all function keys together.

The function keys can be defined from within a program, too. For details, please refer to section D of the system's manual.

Another matter to mention: The => $KEYDEFS utility allows you to store your personal key definitions in a disk file and load them from there when you need them again. How this is done, is explained at the decription of $KEYDEFS.

****
* The Keyboard *
****

A.9.2. Function Key Predefinitions

Most function keys are predefined by the system:

Key	Definition	Function
fixed definition:		
<CMD>	1Bh = <ESC>	Lead-in for screen control codes
<ENTER>	ØDh = <CR>	Pass line to EOS.
re-programmable:		
<TAB>	Ø9h = <TAB>	Horizontal tabulator
<INS>	ØFh = <Ctrl-O>	Insert character
<DEL>	7Fh = <RUBOUT>	Delete character
<DEB>	1Fh = <Ctrl-Underln>	Monitor mode on/off
<BRK>	13h = <Ctrl-S>	Suspend screen output
<Ctrl-BRK>	Ø3h = <Ctrl-C>	Abort program
<UP>	ØBh = <Ctrl-K>	Cursor one line up
<DOWN>	16h = <Ctrl-V>	Cursor one line down
<LEFT>	Ø8h = <Ctrl-H>	Cursor one character to the left
<RIGHT>	ØCh = <Ctrl-L>	Cursor one character to the right
<HOME>	1Eh = <Ctrl-^>	Cursor to Ø,Ø (home position)
<Ctrl-HOME>	1Ah = <Ctrl-Z>	Clear screen
<CL>	18h = <Ctrl-X>	Erase input line
<-ENTER>	<ESC> <'T'>	Erase to end-of-line
<ØØ>	<'Ø'> <'Ø'>	Double Zero
<.>	<'.'>	Decimal point

When re-defining the cursor (arrow) keys, you should remember that the built-in EOS screen editor needs exactly the definitions as specified above for proper function.

Section B

How to Enter Commands

Command Input

The principal task of EOS is to recognize input from the user as meaningful commands and execute them. This job is done by a special program called SHELL. Whenever you are able to enter a command or select a program, you are communicating with this program. Or, to use different terms: When there is no other program running, SHELL is active.

Usually SHELL displays a specific character to signalize that a command is expected, and then waits for input. This input is then interpreted according to definite rules. If a command has been recognized, the corresponding program is searched for, loaded, and control is passed to it.

This kind of entering commands, however, requires some knowledge on the EOS system philosophy. In order to meet the reqirements of all users, especially of those who do not want to work with EOS as an operating system but with customized programs, two different versions of the SHELL program have been developed. The version delivered with your system disk is called SHELL.SYS, which accepts and interprets commands as usual. The other version, called USRSHELL.SYS, represents a total different approach to starting programs: it offers menus, from which the desired program is then selected by keystrokes. There is no simpler way to go. To activate this program as command processor, the file SHELL.SYS has to be renamed, any arbitrary name would suffice (It may be erased, however, this should not be done on the original system disk!). Afterwards, USRSHELL.SYS is renamed to SHELL.SYS. On the next cold start, you will then meet the menu program instead of the command interpreter.

The menu program must be adapted to the application programs it has to control. In most cases, this job will be done by the specialist who installs your programs, he will adapt the menu program accordingly. As an "ordinary" user, you will hardly ever be opposed to this job and may immediately turn to the next chapter. Users who want to make use of the command interpreter, however, may wish to read the subsequent paragraphs.

```
******************************
*     Command Interpreter    *
******************************
```

B.1. The Command Interpreter

The first time you insert your EOS system disk into the drive and switch the machine on, you will see a "greater-than" character on the screen before the computer calms down after the system is loaded and the lights at the disk drives go out. This character is the "ready" prompt of EOS. It signalizes that the command interpreter is now expecting a command from you. For still better clarity, the text "E O S" appears in the leftmost field of the status line. You may now enter any text you like. The interpreter then tries to interpret this as a command. In the subsequent chapters we are going to discuss how the interpreter works. One remark in advance: When executing commands, several ">" characters in sequence appear according to the nesting level of $DO. For details, please refer to the description of the $DO program.

B.1.1. The Screen Editor

Before we discuss details of the command interpreter, just press one of the arrow keys on the righthand side of the keyboard. You will see that you can move the cursor to any desired place on the screen. You may write something nice on the screen if you like.

Of course, there is a well-thought background with it: When you enter a command but make a mistake, you are able to correct the mistype with the aid of the arrow keys. Two further keys serve for such changes, both of them located near the upper right edge of the keyboard. One of them, the "DEL" key, erases the character just under the cursor. The other one is the "INS" key which has the opposite effect: it inserts a blank at the cursor position so you can make room for input corrections.

When you type a whole lot of characters indiscriminatedly, you will see that you cannot write at the leftmost screen column. The reason is that the system prompt ">" appears here, which has to remain and therefore, must not be overwritten.

The "ENTER" key has a special function. When you press "ENTER", the whole story comes to an end. The command interpreter then accepts the line where the cursor is currently positioned, and tries to interpret it as a command (disregarding, of course, the first character which you cannot change anyway).

```
******************************
*    Command Interpretation  *
******************************
```

B.1.2. Input Interpretation

Anytime you have typed a line of text and terminated it by pressing "ENTER", you have entered a command (regardless how meaningful or not your input is). The interpreter turns towards the first word which is always regarded as the command word (There are some special cases we will enter into later on). Let us assume that you have typed:

```
>Oh what a nice computer!
```

In this case, the first word - the command word - is the word "Oh". What now does the computer do with the input line? Fist of all, the entire line is converted to uppercase letters:

```
OH WHAT A NICE COMPUTER!
```

Then, the first word is inspected more closely. The interpreter simply assumes that it is the name of a program file. As the file type is missing, a ".COM" type is appended by default:

```
OH.COM
```

This file is searched for. If it could be found, it is loaded and executed. Furthermore, the remainder of the command line is passed to the "OH.COM" program:

```
WHAT A NICE COMPUTER!
```

"OH.COM" could interpret the remainder of the line by itself. For example, it could create the file "WHAT" and fill it with nothing but "ALAS" or do some other nonsense.

But what if there is no "OH.COM" file at all? The interpreter has not yet reached its limits. It tries another file type, namely ".SUB". If there is a file "OH.SUB", the interpreter assumes that it contains a sequence of commands all of which are to be executed. It therefore starts the program "$DO.COM" which pre-processes the file for interpretation:

```
$DO OH WHAT A NICE COMPUTER!
```

For details on the program "$DO.COM", please refer to the corresponding chapter. At this moment, it is more interesting for us that after we specified the word "Oh", two files are searched for:

first: OH.COM , and second: OH.SUB.

```
******************************
*    Command Interpretation  *
******************************
```

If neither file is present (not too unlikely so), the interpreter is at the end of its capabilities, and it signalizes:

```
Program not found: OH
```

But another thing could happen: imagine that the "OH.SUB" is present, but not so the program $DO which is necessary to interpret the "OH.SUB" file. In this case, the message reads:

```
File not executable: OH
```

To summarize: The first word of a command line is regarded as a filename. If no file type has been specified, the interpreter automatically searches for specific file types. As an extension to this mechanism, you may as well specify a full filename as the first word of a command line, for example:

```
B:OH.YES WHAT A NICE COMPUTER!
```

In this case, the file OH.YES is searched for explicitly on drive B: (and loaded, if present there).

B.1.3. Entering Lowercase Letters

There is a number of tricks you may use when entering commands. Maybe you feel angry about the fact that all input is converted to uppercase letters. If for some stupid accident, a file has been created the name of which contains lowercase letters, you are not able to erase it! But do not surrender, you can get rid of it, of course! Just enclose everything that is not to be converted to uppercase letters in quotation marks. For example: There is a file "JunkYard.DAT" you would like to erase. The command:

```
ERASE JunkYard.DAT
```

will not be successful, there is (hopefully) no file "JUNKYARD.DAT". You have to enclose the filename in quotation marks:

```
ERASE "JunkYard.DAT"
```

and there you are. Later on, we will discuss the ERASE command in detail.

****
* Command Interpretation *
****

B.1.4. Special Characters at the Beginning of a Line

There is a number of special characters a command line may begin with. These characters must be at the very beginning of the line, without any leading spaces. The following special characters are permitted:

B.1.4.1. The Semicolon

A semicolon denotes the beginning of a comment line. The remainder of a comment line is ignored by the command interpreter. This feature is of no use in practial operation, but very meaningful in executable files, i.e. in files containing executable commands.

B.1.4.2. The Colon

A line which begins with a colon is a so-called conditional command line. It is interpreted only if the most recent command could be executed error-free. If, on the contrary, the previous program was terminated due to an error (e.g., an EOS system error), such a line is not interpreted but regarded as a comment line.

The program error code is set by an internal system function. Most currently available programs do not know this new function; hence, it is possible that a program is terminated due to an error condition (recognized by itself), but a conditional command line is executed in spite of the error.

B.1.4.3. The Asterisk

A line beginning with an asterisk is passed directly to the built-in graphics interpreter. It is thus possible to enter graphics commands directly to the command interpreter and interactively draw little pictures. The graphics commands are described in a section of their own.

```
******************************
*         File Search        *
******************************
```

B.1.5. File Search Procedure

Normally, files are searched for only on the current drive and in the current user area. This can be very troublesome, as it would be necessary to have copies of the programs needed on all drives and in each user area. Therefore, the file search algorithm of EOS has been extended.

Let us first talk about user areas. If you are in another user area than user area Ø, none of the commands can be found any longer. In order to have them nevertheless available, files can be marked with a special attribute. You already have learned what file attributes are like. The attribute needed here is the "SYSTEM" attribute. Files with a "SYSTEM" attribute can be accessed from within any other user area. Another advantage: SYSTEM files are not listed in the drive directory unless specified explicitly, thus enhancing the readability of the directory listing. The => $SET utility serves for setting and resetting file attributes.

Furthermore, the command search has been extended to include more than one drive. By means of the => $CONFIG utility, you may define up to four drives which are to be scanned for a command. In its delivery state, EOS automatically begins the seek sequence on the RAM disk, then continues on the current drive. This may be redefined and extended at will.

It is not necessary for the seek procedure that a disk is in each drive specified. If there is no data media in a drive to be examined, it is skipped automatically.

B.1.6. Multiple Command Lines

You may enter more than one command in one command line by seperating these commands by quotation marks "!". To get the directory of a disk, to erase all .BAK files and to copy the remaining files you may enter

```
>DIR -P ! ERA *.BAK ! $COPY *.* B:
```

The program $DO is necessary to interpret such multiple command lines. It would convert the above example into

```
>DIR -P
>ERA *.BAK
>$COPY *.* B:
```

For further details, please refer to the chapter about the program $DO.

****
* Built-in Commands *
****

B.1.7. Built-in Commands

A number of commands are firmly installed in the command interpreter. For reasons of CP/M compatibility, some commands are translated to other command names. In the subsequent chapter, we will discuss these conventions in detail.

A special format of built-in commands is changing the current drive. For this purpose, you have to specify the drive name, followed by a colon:

```
B:
```

for example, defines drive B: as the current drive. The current drive is indicated in the second field of the system status line.

The commands listed below are firmly installed:

```
USER    - Change user area
LIST    - Switch printer on / off
```

The following commands are translated:

```
DIR, DIRS or DIRSYS     - to $DIR
ERA or ERASE            - to $ERASE
TYP or TYPE             - to $TYPE
REN or RENAME           - to $RENAME
```

The utility programs as listed above and their tasks are:

```
DIR                 - Display directory
DIRS, DIRSYS        - Display directory of system files, has the
                      same effect as "DIR +SYSTEM"
ERA, ERASE          - Erase files
REN, RENAME         - Rename files
TYP, TYPE           - Display file contents
```

For details on the application of utilities, please refer to the corresponding chapters in section C of this manual. E.g., if you want to know precisely how to obtain a directory listing, you would read the description of $DIR. At this place, we will give a brief description of the built-in commands.

```
*****************************
*      Built-in Commands    *
*****************************
```

B.1.7.1. USER - Change User Area

The current user area is indicated at the third field of the system status line. It can be interrogated by simply typing "USER":

```
USER

Current user area: 0
```

If you append a number between 0 and 15 to USER, the user area is changed according to the number:

```
USER 15

Current user area: 15
```

If you enter any nonsense, the message reads:

```
Only user areas between 0 and 15 permitted!
```

For CP/M compatibility, the commands "USER" and "USE" have the same effect.

B.1.7.2. LIST - Switch Printer on / off

You may operate a printer parallel to the CRT screen. This command has therefore been implemented. To switch the printer on, you would enter:

```
LIST ON
Printer is switched on.
```

And here is how to switch the printer off (you will have guessed it):

```
LIST OFF
Printer is switched off.
```

If you did a mistype, the message appears:

```
ON or OFF expected.
```

```
******************************
*        Menu Program        *
******************************
```

B.2. The Menu Program

The menu program opens a completely different way of entering commands. When you are using the USRSHELL program, the CRT screen is split into two halves. In the left part, a number of items to select from is listed one upon another, whereas the right half is initially empty. One of the lines in the left part is enhanced by inverse video mode.

Press the up-arrow and down-arrow keys. You will see that you can move the inverse-video field or "window" up and down using these keys. When you stop typing for a while, a text will appear on the right half of the screen giving further explanations on the menu item currently selected by the window.

When a menu item is "lit up" which is of interest to you, press the "ENTER" key. Pressing "ENTER" can have different effects. Normally, a program is started. However, it is as well possible that another menu is activated. Menus can be very deeply nested. To leave a menu, just press the "HOME" key, and the menu program will immediately return to the previous menu.

When you start a program, further specifications may possibly be required. In this case, a number of fields appears in the right half of the screen. Each of the fields contains either a heading or maybe some text. You may alter the text or leave it as it is, according to your wishes. Input fields are marked by underlines. Using the cursor keys, you can jump to any of these fields. The "DEL" key serves for deleting a character. After you have filled a field completely, press the "ENTER" key. The program is started after you have left the last field. At this moment, the screen is cleared and the message

```
Loading XXX.COM
```

appears as a protocol. The message merely indicates that, according to your specifications, a program has been found and loaded. If, instead of, a command file has been activated, the message reads correspondingly:

```
Loading $DO.COM
```

Henceforth, the commands currently executed are indicated as follows:

```
Command: $DIR *.COM
```

In this example, the "$DIR *.COM" command is just executed.

If the menu program calls for additional input from you, but you are currently not willing to meet the program's wishes, just press the "HOME" key, and the program returns to menu items selection mode.

****
* Menu Program *
****

B.2.1. Input with the Menu Program

The table below gives a summary of the input facilities available with the menu program:

Up-arrow	- inverse "window" one line up
Down-arrow	- inverse "window" one line down
ENTER	- select menu item marked by "window"
HOME	- leave current menu

When entering additional parameters, following keys can be used, too:

Left-arrow - Move cursor one character to the left. If the cursor is positioned at the first place of a field, it jumps to the last character of the previous field.

Right-arrow - Move cursor one character to the right. If the cursor is positioned at the rightmost place of a field, it jumps to the first place of the subsequent field.

DEL - Erase the character just under the cursor. If the cursor is at the end of an input line, the character left of the cursor is erased.

HOME - Abort input of additional parameters and return to menu items selection mode.

ENTER - Leave the current input field and jump to the next one. If the cursor was in the last field, the program is started.

```
*******************************
*          Menu Files         *
*******************************
```

B.2.2. Programming the Menu Program

Arriving here, you, honourable user, may have a cup of tea and proceed to the next section. You, dear programmer, would better stay with us for a while.

The menu program is controlled by menu files. Menu files are nothing else but ordinary text files with a ".MNU" file type. At the EOS cold start, the file "EOS.MNU" is loaded as the initial menu.

A menu file is structured as follows: The first column of each line contains a control character specifying the kind of line. There is quite a number of control characters:

H - Header. The subsequent text is displayed centered in the upper left field. Up to 25 characters are taken over.

M - Menu item. Up to 25 characters of the following text are used as a menu item. All subsequent specifications before the next menu item refer to this menu item.

: - Auxiliary text. Per menu item, up to 14 lines of explanations can be specified. A maximum of 50 characters per line will be displayed. The auxiliary text will appear automatically on the CRT screen, if for some 3 seconds, no user input occurs.

S - Submenu. If the menu item has been selected, a submenu is to be initiated. The following text is interpreted as the name of another menu file. If the file type misses, ".MNU" is appended by default.

P - Start program. If the corresponding menu item has been selected, the program denoted by the subsequent text shall be started. The search sequence follows the same scheme as described at the command interpreter. The text may contain an entire command line of up to 80 characters.

E - Enter program parameters. If the command line specified with option "P" is incomplete, it can be completed by the user. User input is appended to the "P" lines in the "E" line sequence before the program is started. The remainder of the "E" lines is divided into two parts by a comma. The first part is displayed as a prompt, whereas the second part serves as a default input. The second part may be omitted.

****
* Menu Files *
****

C - Confirm. After the program run has terminated, a message is displayed in the status line, requesting the user to press the spacebar at the keyboard. This option causes CRT output to be retained before the menu program displays its next menu mask.

; - Comment line.

Any other characters at the beginning of a line cause the remainder of the line to be ignored. It is thus possible to intersperse blank lines in the menu file to enhance its readability.

In order to keep the menu program as small as feasible, the number of syntax checks on the menu file has been kept small. So you must not wonder about menu program crashes due to a badly structured menu file!

B.2.2.1. Menu File Example

Here is an example of a menu file which generates a test menu with three items. The first item causes a submenu to be started, the second one starts a program. The third one also starts a program, but requires additional parameters.

```
; Menu File of a test menu
HTest Menu
; first menu item: start of a submenu
MLoad Submenu
; Help text:
:By selecting this item, you will start
:a great submenu called "NEWMENU.MNU"
; Specify submenu:
SNEWMENU
; second menu item: start of a program
MEdit TESTFILE
:Here, you can edit the "TESTFILE" file.
; Specify start:
PEDIT TESTFILE
; third menu item: start of a program with parameters
MEdit a file
:Here, you can edit a file of your choice
; Specify start:
PEDIT
; Enter filename, default is "TESTFILE"
EPlease enter filename,TESTFILE
; Enter some obscure options:
EPlease enter options as required:
```

Section C

Utility Programs

```
*******************************
*       Utility Programs      *
*******************************
```

C. Utility Programs

In order to make full use of EOS, you need something more than the bare operating system itself: You need a number of small auxiliary routines for those little every-day jobs, as there are erasure and renaming of files, initializing diskettes, and the like. All these routines are called "utility programs" or for short, "utilities".

All EOS utilities have one feature in common: their names all begin with a dollar sign "$". But this is where the similarity ends: the utilities can be classified into two groups. The members of the first group are simply called by typing their name. Then, they display a nice screen mask with a menu to select the desired function from. On the contrary, the others are started by typing a complete command line. Should the command line miss or contain errors, a brief explanation is written on the screen showing how the command line should look like. In the oncoming chapters, we are going to do a closer look on both these kinds of utilities.

C.1. Menu Programs

The first category of utilities are the so-called menu programs. You call them by just typing their name. They then appear with an auxiliary menu on the screen. The most essential feature of a menu program is that you can do all your input with the arrow keys between the alphanumeric keyboard and the numerical keypad. You see that in most cases, one line of the menu is emphasized by inverted video mode. We shall call this line the "window". Using the arrow keys, you may move the window to any desired line. Just try!

When a line is lit up which is of interest to you, press the "ENTER" key. In most cases, you will select one item from a list of several by typing "ENTER". You may also activate a command this way. Commands usually appear on the rightmost side of the menu and are written in capital letters. Presumably, the command most often used is the "QUIT" command, by means of which you leave the program.

There is another way to leave a program: simply press CONTROL-C or the other BREAK character you have defined by means of the => $CONFIG utility. In most cases, this will abort the program.

```
*********************************
*       Utility Programs        *
*********************************
```

C.2. Programs with a Command Line

The other kind of utilities looks different: You invoke them by typing a complete command line. If you are using the SHELL menu program, those are the programs which require additional parameters besides selecting items from the menu. A typical command input could look like this:

```
$SET *.* +SYSTEM
```

C.2.1. Options

Many of the "command line" programs may further be controlled by additional options. Options are specific command words beginning either with a plus "+" or a minus "-" character. However, it is not necessary to specify options, as the name implies. If an option should miss, a default is selected automatically.

Options may appear anywhere within a command line (except before the command name, i.e. as the first word). You need not type the entire name of the option, the first character is fully sufficient. For example, the command:

```
$SET *.* +SYSTEM + BACKUP          has the same effect as
$SET +S *.* +B
```

What do the plus and minus signs mean? Well, options can be activated, or the can be disactivated. If you type a plus, the corresponding option is validated, if you type a minus, it is invalidated. The various programs have different defaults for their options. By default, an option can be "switched on", or it can be switched off. More details are to be found in the following chapters.

****
* $CONFIG *
****

C.3. $CONFIG - Operating System Configuration

$CONFIG allows you to adjust the following parameters of your EOS operating system:

- the number of mini disk drives
- the number of standard disk drives
- the seek sequence of the command interpreter (see section B)
- the drive to be used for temporary files
- the second BREAK character (see section A)
- the tabulator stops of the system
- the system kernel version
- the current year to set the clock correctly

The program itself is neither a menu program, nor can it be called by a complex command line. Rather, after typing the command

```
$CONFIG
```

there begins a question and answer sequence you can run through.

For proper operation, $CONFIG must have access to a system file named "SYSVAR.SYS". This file is always searched for on the disk in the right mini disk drive. $CONFIG waits until you have inserted the correct diskette (normally, the system loading diskette), or you terminate the program by a BREAK character. This file is then changed by $CONFIG and, if you request so, written back. The changes you made are valid from the next system load process, i. e. the next time you switch on the computer.

```
****************
*    $COPY     *
****************
```

C.4. $COPY - The File copy program

As modest as it may appear, this program is certainly the most essential utility of your operating system. It can move your data here and there, and generate backup or working copies of your files. Furthermore, it can address the character I/O devices. You can have $COPY make your computer act as output screen for another computer. When necessary, you can even perform data communications to other computers!

All this is done by a program which even lacks a menu. On the contrary, the only way to activate it is by means of a command line. This is not done to tease you, moreover, $COPY has such a large number of options that it were hardly possible to control all of them by a menu. The options are so numerous that they occupy a considerable part of the alphabet.

C.4.1. Start and Command Input

Let's see how to start $COPY. It is simplicity by itself - you just type

```
$COPY                and immediately, the program responds:

EOS V3 File copy program  (? = Help)

*
```

The asterisk is the prompt of $COPY. Now you can enter your commands to your heart's contents. To begin with the simplest facts: In the first line $COPY displays, it tells you that there is some help at hand. So just type a question mark and press the ENTER key. Promptly, $COPY responds with a list of the options available:

```
Command format:
* source destination options     or:
* destination = source options
pressing ENTER or a stop character end the program.
-ASCII      do not stop transmitting at Ctrl-Z
+BACKUP     backup mode
+CONFIRM    confirm each file
+DEST n     copy to user 'n'
+ERASE      erase source after copy
-INTERRUPT  turn off keyboard checks
-PROT       no console protocol
+REMOVE     remove dest file before copy
+SOURCE n   copy from user 'n'
-TEMP       exclude .$$$ and .BAK files
+VERIFY     verify file copy
+ZERO       zero the most sinificant bit
```

****
* $COPY *
****

A long list, indeed. Let's do a look at the first line:

```
Command format:
* source destination options     or:
* destination = source options
pressing ENTER or a stop character end the program.
```

This is the approximate format of a command line. A command line must at least contain two file designations: the first denotes the source, the second one the destination. For example, if you wanted to copy the file "$COPY.COM" from A: to B:, your command line would look like:

```
* A:$COPY.COM B:$COPY.COM
```

If, on the destintion drive, the file shall have the same name as on the source drive, you need not fully define the second filename. It suffices to specify the drive:

```
* A:$COPY.COM B:
```

If you are already on drive A: (which means that drive A: is indicated in the system status line on the screen), you may as well omit the first drive specification:

```
* $COPY.COM B:
```

That is the shortest way to go. But if you suppose that, when working on drive B:, the command line above might as well look like this:

```
* A:$COPY.COM
```

then you are wrong, because you must anyway specify two file designations.

For those users who are used to the CP/M PIP utility, a second command format is allowed: Enter the destination file name first, followed by an equation character "=" and then by the source file name:

```
* B: = A:$COPY.COM
```

****
* $COPY *
****

Let us go a step ahead now. You will certainly want to copy several files at once. No problem at all: You are free to specify ambiguous filenames both with source and destination. So if you want to copy all ".COM" files from A: to B:, you would type:

```
* A:*.COM B:
```

As simple is copying an entire disk:

```
* A: B:
```

As you may specify ambiguous filenames both with source and destination, a curious side effect results: the files are renamed one by one. For instance, if you should decide to attach a new file type ".OLD" to all ".COM" files on the destination (for whatever reason), the command line looks like this:

```
* A:*.COM B:*.OLD
```

Of course, you can as well copy single files in such a manner that they get new names on the destination drive, for example:

```
* A:$COPY.COM B:$COPY.OLD
```

Again, there is a restriction: When you specify an ambiguous name for the source file, but an unambiguous name for the destination, $COPY is on strike. This would mean that all files are to receive the same name on the destination drive, thus leaving the file transferred last in noble solitude.

To recall the conditions a command line must satisfy in order to be accepted by $COPY:

- always specify two file designations
- Source and/or destination can be ambiguous
- To copy all files of a drive, the drive name suffices
- An ambiguous filename for the source together with an unambigous name for the destination makes no sense and is not permitted.

****
* $COPY *
****

C.4.2. Direct Start

Whenever all copying you intend could be done by typing only one command line, you may append this line immediately to the program name. E.g., to copy all ".COM" files from A: to B:, you only need type:

```
$COPY A:*.COM B:
```

$COPY then replies:

```
EOS V3 File copy program

A:PROGM1  .COM ...
```

The ellipsis (...) stands for the copying protocol we will discuss soon.

C.4.3. The Copying Process and Error Handling

Let us stay with the example of copying all ".COM" files from A: to B:. $COPY then would produce a copying protocol which might look like this:

```
$COPY A:*.COM B:

EOS V3 File copy program

  4 Files found.
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .COM
A:PROGM2  .COM     8 KB  copied to B:PROGM2  .COM
A:PROGM3  .COM    19 KB  copied to B:PROGM3  .COM
A:PROGM4  .COM     2 KB  copied to B:PROGM4  .COM
```

As you can see, $COPY indicates with each file transferred:

- the file currently processed
- its size
- the name of the destination file

Especially the latter feature is advantageous when copying with renaming, as you immediately see the names of the destination files.

****
* $COPY *
****

Here is an eaxmple of copying with renaming to illustrate the effect mentioned on the previous page:

```
$COPY A:*.COM B:$*.OLD

EOS V3 File copy program

  4 Files found.
A:PROGM1  .COM    10 KB  copied to B:$ROGM1  .OLD
A:PROGM2  .COM     8 KB  copied to B:$ROGM2  .OLD
A:PROGM3  .COM    19 KB  copied to B:$ROGM3  .OLD
A:PROGM4  .COM     2 KB  copied to B:$ROGM4  .OLD
```

If it could not find a file as specified, $COPY responds

```
No files found.
```

The following paragraph is devoted to definitely less pleasing matters, namely to errors. There are numerous ways to make errors. When an error is encountered during copying, this is flagged as follows:

```
$COPY A:*.COM B:$*.OLD

  4 Files found.
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .OLD
A:PROGM2  .COM     8 KB  copied to B:PROGM2  .OLD
A:PROGM3  .COM    19 KB  Read/write error.
```

Although the error is indicated after the source file name, this must not necessarily mean that the error occurred when reading the source file. The destination file can as well be affected. In most cases, copying is aborted when an error has been detected.

****
* $COPY *
****

C.4.4. Lack of Space Procedure

More often than one would like a disk suddenly becomes full during copying. Just imagine you want to save the contents of a 5 Megabyte harddisk to diskettes of 390 Kilobytes each, and there you are. In such cases, $COPY tries to do its job with utmost care. Before transferring a file $COPY checks the destination disk if there is enough room to store the file. If space is exhausted, $COPY inspects the destination for a file of the same name already being existing, and if it finds one, it further checks whether sufficient space could be obtained by erasing this file. If all that is true, you are asked if you admit to do so:

```
A:PROGM3  .COM    19 KB  Disk full. Erase destination? (Y/N)
```

If you type "Y" at this moment, the copy protocol reads:

```
B:PROGM3  .COM           erased.
A:PROGM3  .COM    19 KB  copied to B:PROGM3  .COM
```

If there is really no chance to gain more space on disk, $COPY asks:

```
A:PROGM3  .COM    19 KB  Disk full. Change disk? (Y/N)
```

You may now insert a new diskette and continue. If you respond with "N" to both questions, copying is aborted. $COPY also aborts if a question to change data media is superfluous, because the full destination is the harddisk or the RAM disk. In these cases, you obtain a brief error message:

```
A:PROGM3  .COM    19 KB  Disk full.
```

C.4.5 Options

It is becoming interesting now, as we are going to discuss the options of $COPY. For clarity, we have grouped the explanations into several paragraphs. You already saw how to enter options, but some recapitulation weren't the worst thing to do: Options always begin with a "+" or "-" sign. They can be placed freely anywhere in the command line. It suffices fully to specify the first letter of an option. If an option makes no sense, it is simply ignored.

****
* $COPY *
****

C.4.5.1. Copy with Confirm

When you are not fully sure which files to copy, enter the option "+CONFIRM". Then, $COPY asks everytime before transferring a file:

```
$COPY A:*.COM B: +C

EOS V3 File copy program

  4 Files found.
A:PROGM1  .COM    10 KB  Copy? (Y/N) Y
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .COM
A:PROGM2  .COM     8 KB  Copy? (Y/N) Y
A:PROGM2  .COM     8 KB  copied to B:PROGM2  .COM
A:PROGM3  .COM    19 KB  Copy? (Y/N) Y
A:PROGM3  .COM    19 KB  copied to B:PROGM3  .COM
A:PROGM4  .COM     2 KB  Copy? (Y/N) Y
A:PROGM4  .COM     2 KB  copied to B:PROGM4  .COM
```

C.4.5.2. Copy with Pre-Erasure

Normally, $COPY works with utmost care. First of all, it creates a temporary file on the destination drive. Only after this has successfully been filled, an old destination file is erased. As a consequence, disk space may become narrow because two versions of a file may temporarily exist. To prevent this, you may activate the option "+REMOVE". In this case, any existing old destination file is erased before copying:

```
$COPY A:*.COM B: +R

EOS V3 File copy program

  4 Files found.
B:PROGM1  .COM    10 KB  erased.
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .COM
B:PROGM2  .COM     8 KB  erased.
A:PROGM2  .COM     8 KB  copied to B:PROGM2  .COM
B:PROGM3  .COM    19 KB  erased.
A:PROGM3  .COM    19 KB  copied to B:PROGM3  .COM
B:PROGM4  .COM     2 KB  erased.
A:PROGM4  .COM     2 KB  copied to B:PROGM4  .COM
```

****
* $COPY *
****

C.4.5.3. Skip over Temporary Files

Normally, each file found would be copied, thus including old versions and temporary files, too. Usually, these files are marked by the file types ".$$$" and ".BAK". The option "-TEMP" allows for skipping them:

```
$COPY A:*.COM B: -T

EOS V3 File copy program

  4 Files found.
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .COM
A:FILE2   .$$$     8 KB  temporary file, not copied.
A:FILE3   .BAK    19 KB  temporary file, not copied.
A:PROGM4  .COM     2 KB  copied to B:PROGM4  .COM
```

C.4.5.4. Erase Source Files after Copying

If you want to clean up your disk by erasing files after they have been copied, you would activate the option "+ERASE":

```
$COPY A:*.COM B: +E

EOS V3 File copy program

  4 Files found.
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .COM
A:PROGM1  .COM    10 KB  erased.
A:PROGM2  .COM     8 KB  copied to B:PROGM2  .COM
A:PROGM2  .COM     8 KB  erased.
A:PROGM3  .COM    19 KB  copied to B:PROGM3  .COM
A:PROGM3  .COM    19 KB  erased.
A:PROGM4  .COM     2 KB  copied to B:PROGM4  .COM
A:PROGM4  .COM     2 KB  erased.
```

C.4.5.5. Copy with Proofreading

For those cases where maximum data security is a must, there exists the option "+VERIFY". When activated, each file will be proofread after it is written.

****
* $COPY *
****

C.4.5.6. Erasing the Most Significant Bit

Some word processors generate files in which bit 7 of a number of characters is set. Under certain circumstances, these files cannot be processed by other programs. But if you specify the option "+ZERO" when copying such a file, the most significant bit of each character is automatically masked off.

C.4.5.7. Switch Off Protocol

This option has been implemented for those calm people who deem $COPY too verbose. When you activate this option by "-PROT", there will be no copy protocol any longer. In case of errors, however, the file affected will be indicated as explained previously.

C.4.5.8. Back-up of Files

This is a very interesting option. It provides for selectively copying only those files which have been altered since the most recent copy process. For this purpose, $COPY evaluates the BACKUP attribute of a file (see also => $SET). If $COPY finds a file without this attribute, this will be copied and afterwards, it gets a BACKUP attribute attached. If a file is changed later on, EOS automatically resets the BACKUP attribute, so the file will be regarded at the next copy run. If you specify the option "+BACKUP", $COPY switches to back-up or data securation mode:

```
$COPY A:*.COM B: +B

EOS V3 File copy program

  4 Files found.
A:PROGM1  .COM    10 KB  copied to B:PROGM1  .COM
A:PROGM2  .COM     8 KB  already saved, not copied.
A:PROGM3  .COM    19 KB  already saved, not copied.
A:PROGM4  .COM     2 KB  copied to B:PROGM4  .COM
```

After copying is finished, all four files have their BACKUP attribute set.

****
* $COPY *
****

C.4.5.9. Copy between User Areas

If you want to make use of user areas, you must be able to transfer files between different user areas. For this purpose, there are two options available: "+SOURCE" and "+DEST". Both of them are followed by a number designating the desired user area, which must be separated from the option by a space character. "+SOURCE" specifies the user area where files are to be read from, whereas "+DEST" designates the user area where the files are to be written to. If you want, for example, to transfer all of your ".COM" files from user area 9 to user area 12, you would specify the command as (note the space between the option name and the user area number!):

```
$COPY A:*.COM A: +SOURCE 9 +DEST 12
```

C.4.6. Copy via Devices

Beside the usual files and drives, you may as well use $COPY for data transfer to and from logical I/O devices. The devices have specific names and are assigned to a pseudo "drive" X: as implemented with $COPY. Following devices are available:

```
X:CONIN   - the keyboard
X:CONOUT  - the CRT screen
X:LSTOUT  - the printer
X:AUXIN   - the input channel of the auxiliary device
X:AUXOUT  - the output channel of the auxiliary device
X:EOF     - sends a Ctrl-Z as end-of-file character
```

Instead of filenames, any of these devices may be specified as well. There is a restriction insofar as the second filename must be unambiguous. Hence, you may copy

- from an input device to a disk file
- from a disk file to an output device
- or from an input device to an output device.

All data transfer is terminated by Ctrl-Z as end-of-file character. When it is necessary, however, output may be continued even after Ctrl-Z. The option "-ASCII" serves for this purpose; when activated, data transfer will not terminate after a Ctrl-Z is encountered. You will appreciate this feature when transferring program files.

****
* $COPY *
****

A few examples: To obtain a listing of the file "TEXT.DAT" directly on the printer, you would type:

```
$COPY TEXT.DAT X:LSTOUT
```

Should you make up your mind to manually create a file, here is the way to go:

```
$COPY X:CONIN TYPEIN
```

Now all input from the keyboard is directly stored in the file "TYPEIN" unless you type Ctrl-Z.

C.4.6.1. Data Communications

This paragraph is a very essential one. By means of $COPY, you can transfer data from one computer to another via a long data line. Connect both computers using modems and their RS-232 interfaces. Use => $DEVICE to assign the serial interface to the AUXIN: and AUXOUT: devices. Then you are ready to enter the command

```
$COPY filename X:AUXOUT
```

to transmit a file, and the machine at the other end of the line will receive by

```
$COPY X:AUXIN filename
```

For the knowledgeable: An XON/XOFF protocol is used with the AUXIN device. When the internal buffer becomes full (usually after 32 KBytes, if there are no function modules loaded), $COPY sends the control character Ctrl-S (XOFF) via the AUXOUT channel. After the buffer contents have been written to disk and $COPY is ready to receive new data, it sends a Ctrl-Q (XON) character. If the serial interface at the transmitter side is also configured to the XON/XOFF -protocol, transmission is thus automatically suspended and resumed again.

C.4.6.2. Sending the End-of-file character

If communications should get interrupted, the receiver will wait endlessly for an end-of-file mark to correctly terminate transfer. You may supply this character explicitly by typing the command

```
$COPY X:EOF X:AUXOUT
```

Of course, you can as well create an empty file by means of the command

```
$COPY X:EOF NULLFILE
```

if you insist to do so.

C.4.6.3. Direct Screen Output

You can use your computer, for example, as a graphics output terminal for another computer. Map the RS - 232 interface to the AUXIN: device (please refer to => $DEVICE) and enter:

```
$COPY X:AUXIN X:CONOUT -PROT -ASCII
```

By "-PROT" the copy protocol is switched off, and the option "-ASCII" prevents the computer to terminate $COPY after Ctrl-Z is encountered.

C.4.7. Abort Copying

Enentually, you will mistype commands and would therefore like to abort the copy program. You can do this anytime by typing a BREAK character. Copying is then immediately aborted, and the protocol reads, for example:

```
A:PROGM1  .COM    10 KB  *** interrupted ***
```

The devices AUXOUT and AUXIN can also be interrupted by entering a BREAK character. This becomes essential when communications should break down. When you type a BREAK character, Ctrl-Z is sent automatically, resp. deposited in the receiver's file.

****
* $COPY *
****

This leads us to the last option of $COPY: interrogating the keyboard costs time, inevitably. If you are sure that communications run unobjectionably, suppressing the interrupt facility by using the option "-INTERRUPT" might result in a higher transfer speed . This option is available for copying files, too. It is then no longer possible to abort copying from the keyboard.

C.4.8. Error Messages

+SOURCE: User number out of range (0-15)
+DEST: User number out of range (0-15)

You specfied an erroneous user number with these options or simply forgot it.

No target name given.

You specified only one file designation.

Verify error.

During proofreading the file, a difference has been detected. Occurs only along with an active "+VERIFY" option.

Directory full.

A new file could not be created because directory space is exhausted. Erase some files or use => $INITDIR to remove the time stamps, if you have defined them.

*** interrupted ***

The copying process has been aborted due to a BREAK character you typed.

Name conatins '?'.

An unambiguous filename is required here.

Illegal device name.

You used an invalid device name along with the drive designation "X:".

****
* $COPY *
****

Source device must be an input device.

You specified an output device as source.

Destination device must be an output device.

You specified an input device as destination.

Multiple sources and single target not allowed.

You specified an ambiguous filename as source file, but an unambiguous one as destination file.

files identical, not copied.

It can happen that a file shall be copied into itself. $COPY simply ignores this file.

Read/write error.

A physical read/write error was encountered.

Device is write protected.

The destination drive is write protected. Remove the write protect label from the diskette and try again.

File is write protected.

The R/O attribute of the destination file has been set by the => SET utility. Use the same program to remove the attribute.

Invalid device.

You specified a non-existent drive.

File already exists.

This should be a very rare error, is it caused by an internal confusion of $COPY. Erase the destination file manually and try again.

```
****************
*   $COPYDSK   *
****************
```

C.5. $COPYDSK - Copying Entire Diskettes

This utility provides a means for fast duplicating entire diskettes. The disks are copied physically, track by track. As this program is menu driven, you simply type

```
$COPYDSK
```

to start it. The screen mask is divided into three columns, the leftmost of them showing a list of available drives. From this list you select the drive where the source disk is in. The middle column shows the same list, and you select the drive for the (empty) destination disk. Finally, the rightmost column contains the commands available with $COPYDSK.

Principally, it is only possible to copy to a disk of exactly the same format as the source disk. From the first, copying a mini disk to a standard disk and vice versa, is precluded. When the program, after it has started, detects that the destination disk format is not in accord with the source disk, copying is aborted, too.

After the program has started, some kind of scale appears on the screen. The scale is composed of dots, where each dot represents a track to be copied. Two such scales are produced, one for the source disk, the other for one the destination. According to the progress of transfer, the dots are replaced by letters:

- s - System track. These tracks might contain parts of the operation system and are not copied.
- R - Read. After each track has been read from the source, an "R" appears in the upper scale.
- W - Write. In the lower scale, a "W" appears after a track has been written to the destination disk.

It is possible to do a single drive copy; just specify the same drive as source and destination. You will then be asked to change the diskettes before the programm proceeds. Note, however, that the format of both disks are not checked. The destination disk MUST have the same format as the source disk. If you do not take extremely care of this fact, unpredictable results will occur!

```
****************
*     $DATE    *
****************
```

C.6. $DATE - Setting the Current Date

This tiny routine sets the date. You will very seldom need it because your computer has a battery-backed clock which generally must be set only once. To start $DATE, you have to type:

```
$DATE mm/dd/yy hh:mm:ss
```

where "mm/dd/yy" represents the date, and "hh:mm:ss" the time. Both items must be entered in the format as specified above. Hence, the date is written as "month, day, year", separated by slashes, and the time assumes the form "hour, minute, second", separated by colons. If one of the parameters should miss or be erroneous, new input is interactively requested. To set the calendar/clock, you may as well enter:

```
$DATE
```

After date and time have been entered correctly, the computer waits until you hit any arbitrary key before it sets the clock, too. Thus you can set the clock as precise as one second.

Example: Set the clock to August 15th, 1983, a quarter past twelve:

```
$DATE 08/15/83 12:15:00
```

Or the same, this time interactively with a mistype (output from EOS boldface):

```
$DATE
Enter the date as mm/dd/yy : 08/15/83
Enter the time as hh:mm:ss : 12.15.00
Enter the time as hh:mm:ss : 12:15:00

Press the space bar to set the time: <space bar>
```

```
****************
*   $DEVICE    *
****************
```

C.7. $DEVICE - Definitions of External Devices

This program lets you configure your CRT screen, keyboard and printer environment. It offers various facilities for the different interfaces and assigns physical devices to the five logical devices which are defined within EOS.

Your MZ-3500 has a number of physical devices built in, which are listed by $DEVICE in the left column of its initial menu. The devices are:

SCREEN	- the CRT screen
KEYBD	- the keyboard
RS232	- the serial interface
CENTRO	- the parallel Centronics interface
CE332P	- the parallel interface for the CE-332P printer
GRAPH	- the graphics interpreter
DUMMY	- a dummy device which will accept any kind of (otherwise) undesired output. On input, DUMMY returns Ctrl-Z.

The second column indicates the I/O modes of the devices:

input	- Input device
output	- Output device
i/o	- Input/Output device

The contents of the third column can be changed by you. Its first three lines can be selected by means of the cursor keys. When you select one of them and press ENTER, you will see that the text changes each time, offering you another option for that device. Following options are at your disposal.

SCREEN: You may choose between the standard ASCII character set or the German character set, including umlauts. This feature is, however only available together with a special character gernerator ROM which is available from SHARP.

KEYBD: Select between upper case and lower case mode. Normally, you would write lower case letters and press the SHIFT key to obtain upper case letters. You may invert this by means of this option, e.g. for entering programs.

RS232: Here you may select from various transfer speeds and protocols. The protocols CTS and XON/XOFF are available.

****
* $DEVICE *
****

The fourth column lets you assign physical to logical devices. Within EOS, there are five physical devices defined:

CONIN:	- Keyboard
CONOUT:	- CRT screen
LSTOUT:	- Printer
AUXIN:	- Auxiliary device, input
AUXOUT:	- Auxiliary device, output

Currently, the auxiliary device should be of minor interest for you. To assign a physical device to, let's say, the printer, you move the inverse video window to the corresponding field using the arrow keys, then you press ENTER until the desired assignment appears.

Finally, you can transfer the device assignment selected into the operating system. For this purpose, the rightmost column offers three options:

SET transfers the assignment into the operating system, where it remains valid until the next cold start.

SAVE permanently stores the assignment in the operating system. For this purpose, the system load diskette must be inserted into the right mini disk drive. $DEVICE then modifies the file "SYSVAR.SYS" on this disk, so that the newly configured system is available immediately on the next cold start.

QUIT terminates the program without changing the current settings.

****
* $DIR *
****

C.8. $DIR - Drive Directory

Over and over again, you will want to see whatever files are rotating on your diskettes, resp. on your harddisk. The program $DIR has been created to come in here. It informs you which files are found on which disk, tells you how large they are, and what attributes they have been assigned. If you desire, you are further told when a file has been created and when it has been accessed most recently.

There are several ways to start the program. With the command interpreter, you would type:

```
DIR       or
DIRSYS    or
DIRS
```

The first format is the normal format of the command, the other ones are extensions. If you are using the menu control program, just select the item "Directory" (or something similar).

What then, are the results after you entered "DIR" or something similar? Well, you obtain a directory of your disk, ordered alphabetically, and with the file sizes appended. The screen image could look like this:

```
Device A:TEST      DSK          08/16/83  15:30      Access/Update stamping
Name     Type  Blcks      KB
TEST1    .DAT     10       2
TEST2    .DAT    200      26
2 file(s), 28 KBytes used, 334 KBytes remaining. System files exist.
```

The headline indicates the the data media name, the current date and time, and the kind of time stamps. How you can have EOS to enter time stamps into the directory, is explained at the $INITDIR utility. At the left, you see a four-column table. The leftmost two cloumns display filenames and file types, respectively. The third column tells something about the file size: the number denotes how many blocks of 128 bytes each belong to that file. If you multiply this number by 128, you obtain the file sizs in units of Kbytes. The final column specifies the number of Kbytes the file actually occupies.

****
* $DIR *
****

You might ask why there is a need for the number of blocks, when you already have the filesize in Kbytes? Well, this is not too simple. In the worst case, a file can waste up to 16 Kbytes of valuable disk space, though it actually consists of but one block. Thus, both numbers are indicated to enable you to recognize such files.

The bottom line tells you how many files have been found, how much room they occupy and, after all, how much space there remains on that disk. You are further informed that more files would properly have had to be indicated, but these files are "hidden" by a "SYSTEM" attribute attached. You can make those files "visible" by activating an option.

If you are not interested in the entire directory, you are free to limit the number of files which will be displayed: When you specify an ambiguous filename, only those files are indicated which match the name specified. For example:

```
DIR *.COM
```

displays all files of the "COM" type.

```
DIR *.*
```

displays all files, having the same effect as

```
DIR
```

C.8.1. $DIR Options

The directory display format as explained above has a certain feature which can sometimes be disadvantageous: the files are listed page by page, and the screen is erased before a new page appears. Besides, you may occasionally want to see the time stamps. For these and some other requirements, $DIR has a number of options. You will already know what options are.

```
****************
*     $DIR     *
****************
```

C.8.1.1. SYSTEM - Display System Files

If you want to see files which have been "hidden" by means of => $SET (i. e., which have their "SYSTEM" Attribute set), specify the "+SYSTEM" option. Then the screen output could look like this:

```
DIR +SYSTEM
Device A:TEST     .DSK          08/16/83  15:30     Access/Update stamping
Name     Type  Blcks      KB
SYSFILE1.COM     10       2
SYSFILE2.COM    200      26
2 file(s), 6 KBytes used, 334 KBytes remaining. Non-System files exist.
```

As you see, the directory display looks quite different as compared to the previous example. Instead of "System files exist", the message now reads "Non-System files exist". The files indicated now would normally be invisible.

If you are using the SHELL command interpreter, the command

DIRS or DIRSYS

has the same effect as

DIR +SYSTEM

C.8.1.2. ALL - Display All Files

We saw how either normal or system files can be indicated. If you want to see both kinds of files at the same time, please activate the option "+ALL". Then, you would obtain a screen image as follows:

```
DIR +ALL
Device A:TEST     .DSK          08/16/83  15:30     Access/Update stamping
Name     Type  Blcks      KB
SYSFILE1.COM     23       4
SYSFILE2.COM      8       2
TEST1    .DAT    10       2
TEST2    .DAT   200      26
4 file(s), 34 KBytes used, 334 KBytes remaining.
```

****
* $DIR *
****

C.8.1.3. PAGE - Page Mode Display

This option is some kind of a "negative option", what means that normally, page mode display is selected by default. Paging, however, can be very disturbing if, for example, you want to see two directories at the same time. For this purpose, a short version has been introduced, too. You select it by specifying the option "-PAGE", which leads to the following display format:

```
DIR -PAGE

TEST1   .DAT   2K  TEST2   .DAT  26K
2 file(s), 28 KBytes used, 344 KBytes remaining. System files exist.
```

Additionally, the true space occupied by the files is indicated.

C.8.1.4. TIME - Display Time Stamps

Until now, we have been discussing informations on the size of files. As you will certainly know, EOS is capable to protocol the instants of file accesses and file alterations (see also => $INITDIR). To get access to these informations, there is a "+TIME" option. Let's do a look at an example:

```
DIR +ALL +TIME
Device A:TEST     .DSK          08/16/83  15:30      Access/Update stamping
Name     Type  Blcks      KB   Last access         Last Update    Attributes
SYSFILE1.COM     23       4   07/31/83 20:20    07/31/83 14:22   sys
SYSFILE2.COM      8       2   08/01/83 10:23                     sys
TEST1    .DAT    10       2   06/14/83 15:55    06/14/83 15:55
TEST2    .DAT   200      26                     08/01/83 19:41
4 file(s), 34 KBytes used, 334 KBytes remaining.
```

There are three additional columns now. Two of them are time stamps. For example, the most recent alteration of the file "SYSFILE1.COM" was July 31st, 1983, at 14:22, and the last time it has been read was the same day, at 20:30. The final column lists the attributes of each file. You can immediately see that the first two files are "SYSTEM" files, which would normally be excluded from a directory display.

Instead of the "Last access" column, a "Created" column can as well appear, according to the directory definition (refer to => $INITDIR). In this case, you are informed on the instant of the file creation.

```
****************
*      $DIR      *
****************
```

C.8.1.5. USERS - Display all User Areas

Let us do another step ahead. So far, we have been inspecting the current user area. To get access to other user areas too, try the "+USERS" option. Then you would obtain the following image:

```
DIR +ALL +USERS
Device A:TEST     . DSK          08/16/83  15:30       Access/Update stamping
Usr Name     Type  Blcks      KB   Last access       Last Update    Attributes
 0  SYSFILE1.COM      23       4  07/31/83 20:20  07/31/83 14:22  sys
 0  SYSFILE2.COM       8       2  08/01/83 10:23                  sys
11  USERN011.          0       0                  06/14/83 23:59
 0  TEST1    .DAT     10       2  06/14/83 15:55  06/14/83 15:55
 0  TEST2    .DAT    200      26                  08/01/83 19:41
5 file(s)  34 KBytes used, 334 KBytes remaining.
```

Suddenly, there appears one more file in the directory display. This file, USERN011, is assigned to user area 11, and it is enpty.

C.8.1.6. LIST - Direct Output to the Printer

If you want a neat listing on paper (a "hardcopy"), specify "+LIST" as the only option. You will then obtain a printed listing in a manner as if you had activated the options "+ALL" and "+USERS", with a form feed and a page header every 64 lines. On the screen, an instruction reads:

```
Press any key to abort listing...
```

If you follow the advice, the message

```
Listing aborted.
```

appears both on the CRT screen and the printer simultaneously, and the printer quits.

****
* $DISKINF *
****

C.9. $DISKINF - Display Disk Formats

Because EOS knows a grand total of 12 different disk formats, it turned out necessary to add a small program which would indicate the proper format of the current disks. Furthermore, this program gives information on the disk capacity and the space remaining. It is simply started by typing:

```
$DISKINF
```

The answer might read as follows:

```
A: Hard disk D510, 5048 KB capacity, 2000 KB free
B: Hard disk D510, 5048 KB capacity, 5016 KB free
C: EOS Format, 390 KB capacity, 286 KB free
D: not ready
P: RAM disk, 48 KB capacity, 19 KB free
```

You can see that all drives defined are displayed, no matter whether disks are inserted or not. In the above example, there is no disk in drive D:. If there is a disk EOS cannot read, this is indicated as follows:

```
D: not readable
```

If you are not interested in the complete list, you may as well specify the drive you want to be examined with the command line:

```
$DISKINF C:

C: EOS Format, 390 KB capacity, 286 KB free
```

In this case, undefined drives are treated like non-ready drives. So, if you should try to ask for the parameters of a non-existent drive O:, the following will happen:

```
$DISKINF O:

O: not ready
```

****
* $DO *
****

C.10. $DO - Programmed Command Execution

This program has been created to assist with job preparation and control, not as much so for the unprepared newcomer. Nevertheless, it is very useful for everyone feeling angry at the necessity to type in the same command sequences over and over again.

$DO provides you with a facility to deposit command sequences in a file, which otherwise would have to be typed in repeatedly, and have EOS execute them directly from the file. This in achieved by redirecting the keyboard input to a temporary file, from which the input is then read instead of the keyboard. After the file is termiated, it is erased, and the keyboard is revived again.

$DO has been written to put some comfort to this matter. It accepts any desired file, provided that it is of the ".SUB" type. $DO then processes the file, replacing some parts of the text, inserting control characters and doing some other things. Finally the file thus prepared is copied to a temporary file, and the keyboard is "switched over".

Calls of $DO may be nested. You can start $DO from within a $DO file. The nesting level is limited only by the memory available.

It shall be mentioned that the initial job of $DO cannot be aborted by pressing a BREAK character, whereas the execution of the commands stored may be stopped as usual.

C.10.1. Start

To have $DO process a file, you type

```
filename parameter1 parameter2 ...
```

$DO is then started automatically, if the file "filename" is of the ".SUB" type. $DO searches for a file "filename.SUB" and prepares it.

As you see, you may specify parameters after the filename. Usually, those are any texts, separated by spaces or commas. If you want to pass a text containing spaces or commas, simply enclose it in apostrophes:

```
$DO name word1 word2 'This is word 3' word4
```

****
* $DO *
****

$DO now inserts these words at certain places in the file. The underlying mechanism is quite simple: All words passed with the command line after the command designation are numbered from Ø upward. The filename thus becomes word Ø, the word "word1" word No.1, and so forth. At the places of the text where one of the parameters is to be inserted, you would specify a dollar sign "$", followed by the number of the word which you want to be placed there. An example: You have a file "TEST.SUB" of following contents:

```
This is the 1st word: $1
This is the 2nd word: $2, and the 3rd one: $3
```

We now assume that you invoke $DO by the command:

```
$DO TEST WORD1 WORD2 'THIS IS WORD 3'
```

Then, $DO would create a temporary file which looks like:

```
This is the 1st word: WORD1
This is the 2nd word: WORD2, and the 3rd one: THIS IS WORD 3
```

Up to 9 words can be passed from the command line to the text by specifying the parameters "$1" to "$9". Additionally, the name of the file just processed can also be inserted by means of specifying "$Ø".

If you need the dollar sign by itself within the text, there are no obstacles. You should specify it twice if a number follows immediately. Besides, it is good practice to duplicate dollar signs within the text, so that they can easily be discerned from parameters.

Let's have a look at some examples:

```
; this works well:
$$SET FILE +SYS
; this works as well:
$SET FILE +SYS
; but here you must specify two dollar signs:
$$12345678
; in the subseqeuent line, "$1" will be subsituted by a parameter:
$12345678
```

```
****************
*      $DO       *
****************
```

C.10.2. Entering Control Characters

To insert control characters into the text, just make use of the same notation EOS uses for displaying control characters: Enter an up-arrow "^", followed by the letter of the control character. A CONTROL-C is represented in the text by the character sequence "^C". The control characters available are "^A" to "^_":

```
^@   - NUL                          ^P   - DLE = Printer on/off
^A   - SOH                          ^Q   - DC1
^B   - STX                          ^R   - DC2
^C   - ETX = Program abort          ^S   - DC3
^D   - EOT                          ^T   - DC4
^E   - ENQ                          ^U   - NAK
^F   - ACK                          ^V   - SYN
^G   - BEL = Beep ("Bell")          ^W   - ETB
^H   - BS  = Backspace              ^X   - CAN
^I   - HT  = Tabulator              ^Y   - EM
^K   - VT                           ^[   - ESC = Escape
^L   - FF  = Form feed              ^\   - FS
^M   - CR  = Carriage return        ^]   - GS
^N   - SO                           ^_   - US
^O   - SI
```

The control characters as below are precluded:

```
^J - Line Feed   = is simply ignored
^Z - End of File = reserved for the end-of-file sentinel
^^ - RS          = reserved to generate "^"
```

The latter feature enables you to enter the up-arrow "^" in the text by typing it twice, just as the dollar sign.

C.10.3. Further Control Characters

$DO knows further characters with a special meaning, however, these must be placed at the beginning of a line. There are four of them:

1) The colon.

   All lines beginning with a colon contain "conditional commands". The command interpreter will not process them if the previous program terminated with an error condition. This kind of error can be either a return code set by the program, an EOS system error, or an abort due to a BREAK character pressed.

****
* $DO *
****

2) The less-than character "<".

   This character explicitly declares a line as program input line. Normally, $DO passes the file contents to the command interpreter only. All programs which require input from the keyboard must further be served manually. In these cases, lines beginning with a less-than character will be passed to the program.

   Some special regulations are valid now: If a program calls for more input than program input lines are present in the file, $DO switches back to the keyboard until the program has terminated. On the contrary, if there are more input lines than the program requests, the command interpreter displays a warning that superfluous input has been ignored. It then resumes processing at the next system command.

3) The question mark.

   If a line begins with a question mark, an input line is expected interactively from the keyboard. This facility is available with both the system mode and the program input mode.

4) The Semicolon.

   This character is used to mark a comment; however, this feature is available with system mode only. Such a line is not permitted as program input.

C.10.4. Execution Protocol

In order to let you know when $DO is active, two kinds of indications appear. First, the text "Execute" is displayed in the fifth field of the status line. Further, the ready prompt of the SHELL program (but not so with the menu program USRSHELL) is modified. For each activce $DO run, an additional greater-than character ">" is displayed. Hence, with a normal $DO run, the prompt looks like:

```
>>
```

If another $DO run is executed from within a $DO run, the prompt has the following format:

```
>>>
```

and so on.

****
* $DO *
****

C.10.5. Entering multiple commands

As mentioned in chapter B, you may enter multiple commands in a single line. This feature, however, is not interpreted by the SHELL command interpreter itself. Merely, the SHELL passes the whole command line to $DO, which itself creates a temporary file containg all commands. The separator used to separate one command from the next is the exclamation mark "!". When $DO is started, it check whether the command line contains one or more "!" characters. If this is the case, every occurence of the "!" character is substituted with a carriage return/line feed character sequence. This leads to a number of records, each of them terminated with the carriage return/line feed sequence, which then are executed just like any ordinary file. The use of this feature inside a command file is more a question of good taste than of necessity. But one thing shall bve mentioned at this place: As you are permitted to enter multiple commands in a single line, you are permitted to enter program input like inside a command file. Just bear in mind that such a command must start with a less-sign "<".

C.10.6. Examples

Let us assume we have a program called TESTPROG which interactively expects some input lines. An empty line will terminate the program. After TESTPROG is completed, we want to enter a command of our choice. Then, TESTPROG shall run once more, if the command we entered could successfully be executed. The corresponding $DO file would be set up as follows (the lines are numbered for reference only):

```
1: ; Example for lines with special characters:
2: TESTPROG
3: <input number 1
4: <
5: ? Please enter a command
6: :TESTPROG
7: <input
```

The lines and their meanings are:

1: This is a comment, which will be displayed as specified
2: TESTPROG program start
3: Input line for TESTPROG, any text
4: An empty line to terminate TESTPROG.
5: Here, a line of input in requested from the console. The text after the question mark is not interpreted and free for comments
6: Conditional start of TESTPROG
7: Input for the conditional start of TESTPROG
8: After the first input line, TESTPROG (started conditionally) requires further input until an empty line is passed. Then, the command interpreter displays this text.

****
* $DO *
****

C.10.7. Error Messages

Following error messages are defined with $DO:

File <name.SUB> not found.

The file "name.SUB" was not present.

File <name>: Directory full.

The directory of the drive for temporary files is exhausted. It was not possible to create another temporary file.

File <name>: Write error.

The disk is full. The temporary file could not be written to completion.

Unable to execute <name>

The function medule overlay refused to accept the file as input. This error should not normally appear.

<Program input ignored>

This warning is displayed by the command interpreter to signalize that program input lines have been left over from the most recent program run.

<Execution aborted>

If you type a BREAK character while $DO is fetching the next input line, $DO is terminated prematurely, and this message is displayed at the CRT screen.

****
* $ERASE *
****

C.11. $ERASE - File Erasure

The program $ERASE helps you erase files you do not need any longer. As all other utilities, is has a number of particular features and can be extended by various options. If you are using the command interpreter, both forms of the command are applicable:

```
ERA       or
ERASE
```

Both notations of the command have the same effect. When you enter nothing else but the command name, $ERASE will explain you precisely how it is to be called. If you specify a filename, for example:

```
ERASE PROGM1.COM
```

$ERASE will try to erase that file. If the file could be found, this is indicated as stated below::

```
A:PROGM1  .COM (   2 KB)     erased.
1 File(s) erased, 2 Kbytes released, 346 Kbytes remaining.
```

You see that you are informed of how much space the file occupied before it has been erased, and how much space was regained by erasure. If the file was not found, the message reads

```
No files found.
```

Of course, you can erase several files at once by specifying an ambiguous filename. This might look as follows:

```
ERASE *.COM
3 files found. Erase? (Y/N) Y
A:$DIR    .COM (  10 KB)     erased.
A:$SET    .COM (   8 KB) r/o not erased.
A:PROGRAMM.COM (  20 KB)     erased.
2 File(s) erased, 30 Kbytes released, 376 Kbytes remaining.
```

For safety, in the case of several files found, $ERASE asks before erasing a particular file whether you want it like that. The file "$SET.COM" could not be erased, because it has been write protected by means of => $SET. However, activating an option would have it made possible to erase it, too. This leads us to the options of $ERASE.

****
* $ERASE *
****

C.11.1. SYSTEM - Erase System Files only

Usually, $ERASE will affect visible files only. Files which have been attached a SYSTEM attribute by means of => $SET are not erased. Nevertheless, activating the "+SYSTEM" option will allow you to access all such files.

C.11.2. ALL - Erase all Files within a User Area.

Use the option "+ALL" to erase both normal and SYSTEM files.

C.11.3. USERS - Erase Files in all User Areas.

The option "+USERS" enables you to extend the erasure process to all user areas. You may combine this option with the options "+SYSTEM" and "+ALL". In this case, $ERASE displays the user area instead of the drive of the files erased:

```
ERASE *.COM +USERS
3 files found. Erase? (Y/N) Y
0:$DIR     .COM (  10 KB)      erased.
0:$SET     .COM (   8 KB) r/o not erased.
1:PROGRAMX.COM (  20 KB)      erased.
2 File(s) erased, 30 Kbytes released, 376 Kbytes remaining.
```

Here, the file "$DIR.COM" in user area 0 and the file "PROGRAMX.COM" in user area 1 are erased.

C.11.4. R/O - Erase also Write Protected Files

In the previous examples we saw that write protected files are precluded from being erased. If, however, you activate the "+R/O" option, write protection is ignored when erasing, which could look like:

```
ERASE *.COM +R/O
3 files found. Erase? (Y/N) Y
A:$DIR     .COM (  10 KB)      erased.
A:$SET     .COM (   8 KB) r/o erased.
A:PROGRAMM.COM (  20 KB)      erased.
3 file(s) erased, 38 Kbytes released, 368 Kbytes remaining.
```

****
* $ERASE *
****

C.11.5. CONFIRM - Selective Erasure

Use the "+COMFIRM" option to erase files selectively. This will $ERASE cause to inquire you at each file if you want this file to be erased or not:

```
ERASE *.COM +CONFIRM
3 files found.
A:$DIR     .COM (  1Ø KB)      Erase? (Y/N) Y
A:$DIR     .COM (  1Ø KB)      erased.
A:$SET     .COM (   8 KB) r/o not erased.
A:PROGRAM .COM (  2Ø KB)      Erase? (Y/N) Y
A:PROGRAM .COM (  2Ø KB)      erased.
2 file(s) erased, 3Ø Kbytes released, 376 Kbytes remaining.
```

C.11.6. DIRECT - Erase without confirm

From the previous examples you see that $ERASE asks for confirm whenever it finds more than one file to erase. As this can be very hindering when executing command files, you can suppress the inquiry by activating the "+DIRECT" option. In this case, all files found are erased without asking if this is correct.

```
****************
*    $INIT     *
****************
```

C.12. $INIT - Initializing Diskettes

Factory-fresh diskettes are delivered either unformatted or formatted according to some standard, depending on the manufacturer. Diskettes must be formatted in some manner in order to be of any use at all. It is thus necessary to initialize diskettes before putting them into use.

$INIT belongs to the class of menu-driven programs. It is started simply by typing

```
$INIT5  for mini diskettes,
$INIT8  for standard diskettes and
$INITHD for the harddisk.
```

The menu has three columns: The left column serves for selecting the drive with the diskette to be initialized, and the middle column offers a list of available diskette formats. Finally, the right column contains the command list.

For those in a hurry a specific format is predefined. You will see that the window is positioned at the "START" command. This means that a touch of the ENTER key will start the formatting process without delay.

After you selected "START", a dot scale appears in the middle of the CRT screen. Each dot represents a track to be formatted. Of course, the scale for mini disks has another length than that for standard disks. After a track has been initialized, the corresponding dot is replaced by an "I". When the formatting phase has run to completion, $INIT does a second pass over all tracks, this time verifying the disk by proofreading. Accordingly, the "I"s are now replaced by "V"s for each successfully verified track.

If the disk has been found in order after formatting and verifying, the program notifies you in the bottom line of the screen. Any errors are reported there, too.

What if you want to stop initialization? Well, it is the standard procedure you will know since long: type CONTROL-C or another => BREAK character. Formatting is immediately aborted.

```
**************************************************************************
*                                                                        *
*         Initializing diskettes will destroy any data stored            *
*                                                                        *
*                          C O M P L E T E L Y  !                        *
*                                                                        *
*              So be careful to insert the correct diskette              *
*                                  BEFORE                                *
*                            you select "START" !                        *
*                                                                        *
**************************************************************************
```

****
* $INIT *
****

C.12.1. Mini Diskette Formats

For mini diskettes, the following formats are defined:

FDOS/CPM - This is the format of the SHARP FDOS operating system. Furthermore, there is an implementation of CP/M 2.2 which uses the same format.

MZ-80B - The format of the SHARP MZ-80B computer. Along with the MZ-80B screen mode availability, this provides for full compatibility of software which has been developed for the MZ-80B CP/M implementation.

PC-3201 - The DiCOS format of the SHARP PC-3201 computer. If you are happy owner of a PC-3201 and DiCOS, you will certainly appreciate it.

MZ-35xx - The EOS standard format of your computer. It combines very high data density with the fastest possible access. It represents the default disk format of the $INIT5 program.

C.12.2. Standard Diskettes

As with mini diskettes, there are four different formats at your disposal. Properly spoken, these are not less than eight formats, as the formats for single-sided disks differ from those of double-sided disks. The formats are:

128 Bytes - The single-sided version of this format has been established worldwide as the standard format for software exchange. It is known by software distributors as "A1" format. The format is recommended for exchanging data and programs.

256 Bytes - The double-sided version is compatible to the HAYAC-2900 by SHARP, whereas the single-sided version is compatible to the format of the IMS-8000 family of computers.

512 Bytes - Just for completion. Not compatible to any other system.

1024 Bytes - The EOS standard format for standard diskettes. It is compatible to the EOS format of the IMS-8000 computer family, being the default of the $INIT8 program.

****
* $INIT *
****

C.12.3. The Harddisk

The $INIT program for the harddisk behaves somewhat different from the corresponding diskette programs. First, the middle column of the menu is omitted, because it is not necessary for you to determine the recording format of the harddisk.

Another deviation concerns the treatment of reading errors during verification. The diskette formatting programs just abort on a proofread error and advise you to deposit the defective diskette. After long and careful consideration, this method has been declared unpracticable with harddisks. Hence, defective blocks are collected into a particular file. This file then occupies exactly those blocks on the harddisk which lead to verify errors, thus locking them for other files.

For people who want to know precisely: The file is named "BadBlock.SYS", allocated in user area 15 and has all three attributes "R/O", "SYSTEM" and "BACKUP" set.

****
* $INITDIR *
****

C.13. $INITDIR - Controlling Time Stamps

EOS has the blessing capability to protocol the moments of file accesses. Following times can be stored for each file:

1. The instant of the most recent access,
2. The instant of the most recent alteration,
3. The instant of the creation of the file.

However, per file, only two of these moments can be protocolled. This means that you have to decide whether you want time stamps of access or time stamps of creation, as these are mutually exclusive.

To make time stamps visible, the "TIME" option of the $DIR command has been created. For details, please refer to the description of $DIR.

The program $INITDIR controls time stamp handling. The program is menu-driven. In this case, no futher explanations are necessary here, as the program is fully self-explanatory. Its call simply reads:

```
$INITDIR
```

In the directory, some space must be reserved for the time stamps. Hence, you will find the item "WRITE STAMPS" in the $INITDIR menu. When you select this item, space is created within the directory to store time stamps. Time stamps consume about a quarter of the directory space available, so when they are activated, you can create 75 percent of the otherwise available file entries. Usually, a directory overflow will not occur. However, if it should happen, you can remove the time stamps from the directory anytime using the command "ERASE STAMPS".

$INITDIR further offers a facility to name the diskettes, i.e. the data media themselves. These names or "labels" are displayed in the upper left corner of the screen when you are inspecting the directory. To create such a disk label, just select the menu item "enter directory label". The window then jumps to the field beneath, where the directory label is displayed. Here you may enter a new label of your choice. To leave the field without entering a new label, simply press the HOME, the cursor-up, or the cursor-down key. If you enter a new label, do not forget to terminate, as usual, your input with ENTER so that the new label is written to disk.

The label can be written separately by selecting the command "WRITE LABEL". You should always do so after having selected the time stamps mode, so that the corresponding control information is written to disk along with the label.

****
* $KEYDEFS *
****

C.14. $KEYDEFS - Storing and Loading Key Definitions

After you have programmed your specific key definitions using the CTRL-7 and CTRL-8 keys (please refer to section A), you will certainly want to preserve your labour for posterity. For this purpose, type:

```
$KEYDEFS +SAVE      or      $KEYDEFS +S
```

This program writes your key definitions to the "KEYDEFS.KEY" file. If you want to restore these definitions, this is simply done as follows:

```
$KEYDEFS
```

As with any other option, you may abbreviate "+SAVE" as "+S".

In the command line, you can further specify the name of a file the key definitions are to be written to, resp. loaded from. If you omit the file type, the type ".KEY" is generated by default. For example, key definition transfer to/from a file called "MYKEYS.KEY" is specified as follows:

```
Store:   $KEYDEFS MYKEYS +S
Load :   $KEYDEFS MYKEYS
```

```
****************
*     $MOD     *
****************
```

C.15. $MOD - Working with Function Modules

This program is more a tool for software designers than a general utility. As an "ordinary" user, you will hardly ever need it. But if you are interested in function module techniques and applications, you may want to read this chapter.

To provide for the highest possible flexibility, EOS can be extended by programmers. This is achieved by so-called function modules. A function module is a specific program written in assembly language, which is then appended to other "ordinary" programs by means of the $MOD utility. If a function module is encountered when loading a program extended this way, it is transferred to the upper end of available memory. It there intercepts all system calls from the main program, interpreting some of the system calls by itself.

The $DO utility is an example of such a function module. The program itself does nothing else but convert the file specified and generate a temporary command execution file. This file is then executed by an overlaid function module, which intercepts EOS function 68 and interprets the commands by itself. Hence, if some other program wants to make use of EOS function 68, it must be guaranteed that the corresponding function module is available.

In most cases, a function module determines by itself how long it remains in memory. After it has done its job, it sets a specific flag. Then, EOS will remove the module from memory after the current program has terminated.

Function modules are stored in files of the ".RSX" type, which is derived from "Resident System Extension". The internal structure of function modules is described in section A of the system manual.

The program $MOD now has been created to append one or more function modules to other programs. Creation and removing of modules can be controlled by a number of options. Without options, there are two ways of calling the program we are going to describe soon.

A peculiarity shall be mentioned: Under EOS, loading programs is done by a function module, too. This module, called "$PGMLDR", is loaded into memory in conjunction with the SHELL command interpreter, because SHELL itself is not capable of loading programs. Usually, the loader is removed from memory before the program loaded is started. However, if further modules are loaded, the loader will remain in memory and can be used by means of EOS function 59. Using a special option, a program can be modidfied in such a way that the loader remains in memory without loading additional function modules.

****
* $MOD *
****

C.15.1. Appending Function Modules

This call has the following format:

```
$MOD comfile rsx1 rsx2 ...rsx15
```

The first parameter "comfile" denotes the ".COM" file to be extended by function modules. It is followed by a list of the RSX files which contain the modules to be appended. A maximum of 15 modules can be specified for a ".COM" file. The modules are loaded in the same sequence as specified in the command line. This has the side-effect that of two modules which intercept the same system call, the one most recently loaded will be active.

If the ".COM" file is already extended by modules, the modules specified new are appended after all yet existing modules. If a module occurring in the list should be present in the ".COM" file, it will be replaced by the new one.

An example: You have a file named "PROG.COM". This file contains the modules "MOD1", "MOD2" and "MOD3" (maybe, besides the main program). Now you enter the command

```
$MOD PROG MOD1 MOD4
```

This causes the module MOD1 in the PROG.COM file to be replaced by the new module MOD1. The module MOD4 is appended at the end of the list. When you execute the program, first the module MOD4 intercepts EOS calls, then MOD3, MOD2 and finally, MOD1.

C.15.2. Removing Function Modules

To remove all function modules from a ".COM" file (thus causing a normal ".COM" file being generated), you would simply omit the module specifications from the command line:

```
$MOD progname
```

By that means, all modules are removed from the file "progname.COM".

C.15.3. LOADER - Keep Program Loader in Memory

If you want to call the program loader from within your program but do not need any function modules, you would type:

```
$MOD progname +LOADER
```

The result is that the file "progname.COM" is provided with a specific header, signalizing the loader that it must not remove itself. Afterwards, the program can make use of system call 59.

C.15.4. COM - Concatenating Function Modules

Sometimes it will happen that you need one or more modules for several programs in sequence. To avoid linking the modules to each program, this option allows you to generate a file which consists of function modules only. The modules may even have their inactivity flag set; they are not removed from memory before the next program run is terminated. For example, if you enter the command:

```
$MOD MOD1 MOD2 MOD3 +COM
```

the modules MOD1.RSX, MOD2.RSX and MOD3.RSX are concatenated to a file named "MOD1.COM". This file can be executed like any other program file and does nothing else but activate the modules contained in it.

C.15.5. PROT - Suppress Protocol

Usually, you would obtain a loading protocol on the CRT screen, which gives you informations on the structure of the file created. The loader protocol could, for example, look like:

```
Name    Type  Offset  Size
PROG    .COM   0100   1000
MOD1    .RSX   1100   02E0   old
MOD2    .RSX   1480   00F0   replaced
MOD3    .RSX   1600   0300   new
```

The column "Offset" specifies at which address relatively to the begin of the file, the corresponding module begins. "Size" denotes the length of the module code in bytes. Furthermore, it is further if the module appended has been preserved from the old ".COM" file ("old"), replaced by a new one, or linked-in new. If you do not want a protocol, activate the "-PROT" option.

```
****************
*      $MOD      *
****************
```

C.15.6. MODULES - List all Active Modules

For completeness, this option has been implemented here, too. When you enter:

```
$MOD +MODULES
```

you will obtain a list of all currently active modules on the screen:

```
Currently active modules:

Name        Strt   Size
$DO         E706   0500
$DO         EC06   0500
$PGMLDR     F106   0300
```

In this example, a two-level nested $DO run is in progress. Therefore, the program loader remains memory.

The option may also be entered in conjunction with an ordinary command line. In this case, the modules are listed before the command is executed. When there is no active module, the message reads:

```
No function modules active.
```

****
* $RENAME *
****

C.16. $RENAME - Alter Filenames

The program $RENAME enables you to alter filenames. As all other utilities, is has a number of particular features and can be extended by various options. With the command interpreter, both forms of the command are applicable:

```
REN      or
RENAME
```

Both notations of the command have the same effect. When you enter nothing else but the command name, $RENAME will explain to you precisely how it is to be called.

There are two ways of using the RENAME command, the first of them being compatible to CP/M:

```
RENAME filename.new = filename.old
```

In this mode, you specify the new filename first, followed by an "equals" sign and the old filename. As many people find this "mathematical" notation somewhat circumstantial, $RENAME allows for another way of calling it:

```
RENAME filename.old filename.new
```

where you specify the old filename first, then the new one without an "equals" sign between the filenames. Choose the mode you prefer.

But that is not all. $RENAME will accept ambiguous filenames as well. The command:

```
RENAME *.COM *.OLD
```

for instance, will rename all your ".COM" files to the ".OLD" type.

Renaming using ambiguous is performed like this: If a file is found with a matching filename, all question marks or asterisks specified in the the RENAME command are substituted by characters from the old filename. The other characters are taken as specified in the command. If, however, this would lead to an invalid filename, no substitution will take place. For example, let us assume you have three files:

```
LONG1111.DAT   SHORT.DAT       LONG2222.DAT
```

```
****************
*    $RENAME   *
****************
```

Then, you enter the command

```
RENAME *.DAT ???????X.DAT
```

which would normally produce the following filenames:

```
LONG111X.DAT   SHORT  X.DAT   LONG222X.DAT
```

You will immediately see that the second name "SHORT X.DAT" is not valid, because it contains space characters. Therefore, the files would be renamed as:

```
LONG111X.DAT   SHORT.DAT       LONG222X.DAT
```

The "X" in the second filename is simply ingnored.

Whenever you are using ambiguous filenames, you should observe that no two files of the same name would be generated; otherwise, $RENAME will refuse to rename the second file.

Example:

```
RENAME *.COM *.CIM
3 files found. Rename? (Y/N) y
A:$DIR    .COM ( 11 KB)     renamed into $DIR    .CIM
A:$SET    .COM (  8 KB) r/o not renamed
A:$ERASE  .COM (  8 KB)     renamed into $ERASE  .CIM
```

For safety, in the case of several files found, $RENAME asks before renaming a particular file whether you want it like that. The file "$SET.COM" could not be renamed, because it has been write protected by means of => $SET. However, activating an option would have it made possible to rename it, too. This leads us to the options of $RENAME.

C.16.1. SYSTEM - Rename System Files only

Usually, $RENAME will affect visible files only. Files which have been attached a SYSTEM attribute by means of => $SET are not renamed. Nevertheless, activating the "+SYSTEM" option will allow you to access all such files.

C.16.2. ALL - Rename all Files within a User Area.

Use the option "+ALL" to rename both normal and SYSTEM files.

****
* $RENAME *
****

C.16.3. USERS - Rename Files in all User Areas.

The option "+USERS" enables you to extend the renaming process to all user areas. You may combine this option with the options "+SYSTEM" and "+ALL". In this case, $RENAME displays the user area instead of the drive of the files renamed:

```
RENAME *.COM *.CIM +USERS
3 files found. Rename? (Y/N) y
 Ø:$DIR     .COM ( 11 KB)      renamed into $DIR     .CIM
 1:$SET     .COM (  8 KB) r/o not renamed
 Ø:$ERASE   .COM (  8 KB)      renamed into $ERASE   .CIM
```

C.16.4. R/O - Rename also Write Protected Files

In the previous examples we saw that write protected files are precluded from being renamed. If, however, you activate the "+R/O" option, write protection is ignored when renaming, which could look like:

```
RENAME *.COM *.CIM +R/O
3 file(s) found. Rename? (Y/N) y
A:$DIR     .COM ( 11 KB)      renamed into $DIR     .CIM
A:$SET     .COM (  8 KB) r/o renamed into $SET     .CIM
A:$ERASE   .COM (  8 KB)      renamed into $ERASE   .CIM
```

C.16.5. CONFIRM - Selective Renaming

Use the "+COMFIRM" option to rename files selectively. This will $RENAME cause to inquire you at each file if you want this file to be renamed or not.

```
RENAME *.COM *.CIM +CONFRIM
3 file(s) found.
A:$DIR     .COM ( 11 KB)      Rename? (Y/N) y
A:$DIR     .COM ( 11 KB)      renamed into $DIR     .CIM
A:$SET     .COM (  8 KB) r/o not renamed
A:$ERASE   .COM (  8 KB)      Rename? (Y/N) y
A:$ERASE   .COM (  8 KB)      renamed into $ERASE   .CIM
```

C.16.6. DIRECT - Rename without confirm

From the previous examples you see that $RENAME asks for confirm whenever it finds more than one file to rename. As this can be very hindering when executing command files, you can suppress the inquiry by activating the "+DIRECT" option. In this case, all files found are renamed without asking if this is correct.

```
****************
*    $SAVE     *
****************
```

C.17. $SAVE - Dump Memory to Disk

This program has a very specific function ordinary users will hardly ever need. It is of interest for programmers only.

Under CP/M 2.2 resp. DiCOS, the SAVE command was firmly installed in the command interpreter. By that means it was possible to transfer parts of the memory contents to a disk file. It is usual practice to load a program into memory using a suitable test program ("debugger"), test and alter the program and with the session finished, to save the program to disk by means of the SAVE command.

EOS does not know this command, as the command interpreter is a program like any other program, thus residing in the same area of memory. For this reason, another mechanism had to be implemented to provide for saving memory contents to disk before memory is overwritten by the command interpreter.

Essentially, $SAVE is nothing else but a function module. If you enter the command

```
$SAVE
```

the module is loaded into memory directly beneath the EOS system entry, where it remains inactive until the next program terminates. After the succeeding program has finished (e.g., a debugger), $SAVE comes in with the message

```
EOS V3 - Memory save utility

File name  :
```

Here you would enter the name of the file to which memory contents are to be transferred. If you just press "ENTER" and nothing else, $SAVE aborts. If the file specified is already present, you are inquired:

```
filename.typ: Erase? (Y/N)
```

If at this moment, you type anything else but "Y", another filename is asked for. The next questions concern the start and end address of the memory area. After the file is written, $SAVE disactivates itself.

****
* $SAVE *
****

Example: You would like to save the memory area from 1ØØH to 3FFH in the file TEST.COM.

```
>$SAVE

><debugger run, or anything else>

EOS V3 - Memory save utility

File name     : test.com
TEST.COM: Erase? (Y/N) y
Start address: Ø1ØØ
End address   : Ø3FF

End <Debugger> 1:ØØ
>
```

The $SAVE.RSX function module belongs to the EOS scope of delivery. Using the => $MOD utility, you may directly link this module to a debugging program, thus becoming releaved from the necessity to start $SAVE explicitly.

Internal structure and operation of function modules is discussed in detail in section A of the System Manual.

```
****************
*    $SET      *
****************
```

C.18. $SET - Set and Reset Attributes

Here and there within this manual, the file attribute "SYSTEM" will have crossed your way. $SET provides the necessary means to set and reset this and other attributes.

Most of the possible attributes are reserved for EOS itself, but three of them can be set by you at will:

SYSTEM - This attribute makes a file globally accessible. It can then be read from within other user areas, too. Furthermore, a file with a SYSTEM attribute is not displayed with the directory of a disk, thus improving the readabilty of the list (in some cases, substantially so).

BACKUP - The => $COPY utility sets this attribute of each file which has been secured. On the other hand, EOS resets the attribute automatically if a file has been altered. Thus, files which have not yet been secured can easily be recognized. A special mode of $COPY only copies files without this attribute.

R/O - Files with an R/O (Read Only) attribute can neither be altered, nor can they be erased.

How to set these attributes? Just specify the desired filenames along with the corresponding attributes, lead in by a plus or minus sign. If you do not specify attributes, they will not be changed.

Let's have a look at some examples. First, you set the SYSTEM attribute for *.COM files:

```
$SET *.COM +SYSTEM
PROG1    .COM (  4 KB) set to sys.
PROG2    .COM ( 10 KB) set to sys.
PROG3    .COM (  1 KB) set to sys.
```

Now, you set the write protection of the PROG2.COM file and reset the SYSTEM attribute:

```
$SET PROG2.COM +R/O -SYSTEM
PROG2    .COM ( 10 KB) set to r/o.
```

Finally, you reset the SYSTEM attribute of all files:

```
$SET *.COM -SYSTEM
PROG1    .COM (  4 KB) reset.
PROG2    .COM ( 10 KB) set to r/o.
PROG3    .COM (  1 KB) reset.
```

```
****************
*    $SPOOL    *
****************
```

C.19. $SPOOL - Background Print Services

Presumably, the $SPOOL utility is one of the most pleasing features of your EOS operating system. The purpose of this program is to print your files at the printer while you are busy doing other jobs. The program is started by a command line. If you want to print a file in background mode, you enter:

```
$SPOOL filename
```

If $SPOOL was able to start printing that file, it notifies you:

```
Printout started.
```

If background print is already active, the message reads accordingly:

```
Spooler already active.
```

If the file to be printed was not found, it reads:

```
File not found.
```

In order to avoid any unnecessary pre-processing of the file, $SPOOL does some additional work. The tab stops found in the file are automatically expanded to the current system tab stop setting. This means that there is a tab stop every eight columns unless you have changed the setting by means of the => $CONFIG utility. Furthermore, $SPOOL inserts a form feed every 64 lines. You may, however, activate an option to suppress form feeds.

There are, almost inevitably, some disadvantages with $SPOOL: operating the printer slows down screen output somewhat. Of course, the printer is no longer available for your current program as long as background printing is in progress. If you want to address the printer directly, either by a program or by the "PRINTER ON" command (please refer to section B), nothing at all will happen. Printer output is simply ignored.

It is self-evident that a print program of this kind cannot do without some means of control. The following options are available with $SPOOL:

****
* $SPOOL *
****

C.19.1. Options

A few words in advance: Besides "PAGE", none of the $SPOOL options requires a filename. If $SPOOL is called along with an option without background printing being in progress, the message is displayed:

```
No spooling active.
```

C.19.1.1. STOP - Suspend and Release Printing

If you want to stop printing, enter the command

```
$SPOOL +STOP
```

$SPOOL then replies:

```
Printout stopped.
```

To continue printing, you would type:

```
$SPOOL -STOP
```

and $SPOOL answers:

```
Printout continues.
```

C.19.1.2. RESTART - Restart Printing

If you decided differently and want to print from the beginning of the file again, you would enter:

```
$SPOOL +RESTART
```

$SPOOL then resets and anwers:

```
Printout restarted.
```

****
* $SPOOL *
****

C.19.1.3. ABORT - Abort Printing

This option serves as an emergency break if for some reasons, the listing does not please you at all. You type:

```
$SPOOL +ABORT
```

and printing is immediately aborted. The program replies with the message

```
Printout aborted.
```

C.19.1.4. PAGE - Suppress Form Feeds

As previously mentioned, $SPOOL is so kind to automatically insert a form feed every 64 lines. This is convenient in most cases, but it can be very disturbing. If you want to renounce form feeds, specify the corresponding option after the filename:

```
$SPOOL filename -PAGE
```

In this case, $SPOOL will not insert form feeds by itself.

C.19.2. Additional Information for Connoisseurs

For everyone who wants to know precisely: The function module $SPOOL.RSX intercepts EOS system calls 1, 2, 5, 9, 10, 11 and 111. Furthermore, the vectors 3, 4, 5 and 6 in the BIOS jump vector table are altered such a way that programs which do console I/O directly via the BIOS jump vector are also affected. All EOS calls, with the exception of system function 10, have a priority less than that of the CRT screen. More precisely, the printer is polled on each eighth CRT output, resp. status check in order not to slow down screen output more than necessary. Serial printers should be operated at a maximum speed of 1200 baud, as this will result in the least loss of screen output speed.

```
****************
*    $TYPE     *
****************
```

C.20. $TYPE - Display File Contents

This little utility lets you display files on the CRT screen, or list them on the printer. It too has a number of options to put comfort to your work. With the command interpreter, you would call the program as follows:

```
TYPE filename options
```

"Filename" denotes the name of the file you wish to see. Options are, as the term implies, optional. If the file could not be found, this is objected by issuing the question:

```
Type which file?
```

You may as well specify a whole bunch of files by entering an ambiguous file name. These files are then displayed in alpabetical order.

This program, too, comes along with some nice options:

C.20.1. PAGE - Page Mode Output

This is the standard output mode. The file contents are displayed on the screen page by page. When you press the space bar, the next page will appear. If you type a BREAK key instead of, output is aborted. You may inactivate this option by specifying "-PAGE". Screen output then runs continuously, and you can suspend it by pressing Ctrl-S, continue by typing Ctrl-Q, or finally, abort by touching a BREAK character.

C.20.2. SLOW - Slow Mode Output

If you specify the "+SLOW" option, output is done continuously but slowly enough so that you can stop at specific places.

****
* $TYPE *
****

C.20.3. CTRL - Handling Control Characters

Normally, control characters found in the text are output directly, so you can intersperse screen or printer control characters in the text. For testing purposes, this kind of interpreting control characters can be switched off using the "-CTRL" option. In this case, control characters are denoted by the character sequence "^x", where x represents the letter associated with the corresponding control character.

C.20.4. LIST - Redirect Output to the Printer

If you want a neat listing on paper (a "hardcopy"), specify "+LIST" as the only option. You will then obtain a printed listing, with a form feed and a page header every 64 lines. On the screen, an instruction reads:

    Press any key to abort listing...

If you follow the advice, the message

    Listing aborted.

appears both on the CRT screen and the printer simultaneously, and the printer quits.

Section D

The Graphics Interpreter

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D. The Graphics Interpreter

As you know, your computer can generate phantastic colour graphics. These graphics, of course, had to be included in EOS in some manner. After careful considering we decided to provide for two ways of accessing graphics. The first way is interesting for system programmers and leads directly into the EOS kernel by EOS function 115. This system function is discussed in detail in section C of the system manual. In this section, we will describe the other graphics facility which can be used by every programmer.

At the beginning of the implementation there came the question of how it would be possible to control graphics by every programming language. To avoid the necessity of linking assembly language subroutines to user programs, a library of graphics functions (which, in practice, would have required several libraries for various programming languages) was precluded. The only remaining way was to use standard interfaces. Hence, a graphics language was developped which is invoked via the CRT interface. Sending certain control sequences to the CRT driver activates the graphics interpreter.

D.1. Activating the Graphics Interpreter

To start the graphics interpreter, there are three modes to select from:

1. Via the CRT interface. When you send the character string <ESCAPE>I (uppercase I) to the CRT, normal screen display is switched off. All screen output is redirected to the graphics interpreter. This mode is terminated at the end of the program run or by sending the graphics command "END".

2. By the configuration program $DEVICE. The graphics interpreter is installed in the operating system as the physical device "GRAPH". You may assign this device firmly to the screen, the printer, or the auxiliary device. All output intended for one of these devices is then processed by the graphics interpreter.

3. By a special command of the SHELL command interpreter. This is more some kind of game and primarily intended for testing the graphics commands. All command input beginning with an asterisk "*" is passed directly to the graphics interpreter. If you like to, you may interactively create some graphics. Of course, this is not possible with the USRSHELL menu program.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.2. Instruction Format

Each graphics instruction consists of a normal line of text, terminated by a <CR><LF> control character pair. This means that you can generate these instructions by ordinary output statements from within your application program. Each line begins with a graphics instruction code written in plain text, most of them being three or four characters long. The character mode is of no concern, as the interpreter will accept uppercase as well as lowercase characters. The instruction code is followed by parameters, if the specific command requires any. The parameters are separated by commas. Space characters may be interspersed freely anywhere in the instruction line (of course, not within the instruction code or a parameter itself).

Each parameter has certain limits. If these bounds are exceeded, the maximum possible value is substituted in most cases. The limits are:

```
X-axis      : Ø to 639
Y-axis      : Ø to 399
Colour      : Ø to 7
Bit images  : Ø to 126
```

If you omit one or more parameters, they are substituted by zero values. So do not be surprised to see nothing on the screen when you accidentially forget to specify the colour, thus "plotting" black (colour = Ø) on a black screen!

Let's have a look at the parameters: Graphics are located in the so-called second quadrant. This means that the coordinate point (Ø,Ø), also called the "origin", is located in the upper left corner of the screen. From here, the x-axis runs to the right edge of the screen, whereas the y-axis runs to the bottom. A line consists of a multiple of 16 image dots or "pixels". Normally, these form a continuous line. But you may redefine the line mode by the graphics instruction "MASK", so that you can also generate dotted or dashed lines.

There are eight different colours available. The list below shows you the colours and their corresponding codes which have to be specified with a number of instructions:

```
Ø - black
1 - blue
2 - red
3 - magenta
4 - green
5 - cyan
6 - yellow
7 - white
```

All other values will be substituted by 7 = white.

This page has been left blank for your notes.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.3. Bit Image Definition

The graphics interpreter offers you a facility to define your own character set. Using the "TEXT" instruction, you can write any desired text on the screen which may contain your own characters. The "FILL" instruction which fills specified areas with some pattern also makes use of this character table , so that you can create your own fill patterns.

Each character must be regarded as a matrix of 8 times 8 bits. The "IMG" instruction lets you define up to 8 bytes of 8 bits each for a character matrix. For example, let's see what the matrix of the letter "A" looks like:

```
Byte 1 :   00011000  =  18H      . . . * * . . .
Byte 2 :   00100100  =  24H      . . * . . * . .
Byte 3 :   01000010  =  42H      . * . . . . * .
Byte 4 :   01111110  =  7EH      . * * * * * * .
Byte 5 :   01000010  =  42H      . * . . . . * .
Byte 6 :   01000010  =  42H      . * . . . . * .
Byte 7 :   01000010  =  42H      . * . . . . * .
Byte 8 :   00000000  =  00H      . . . . . . . .
```

Another kind of pattern can be used to fill areas:

```
Byte 1 :   10001000  =  88H      * . . . * . . .
Byte 2 :   01000100  =  44H      . * . . . * . .
Byte 3 :   00100010  =  22H      . . * . . . * .
Byte 4 :   00010001  =  11H      . . . * . . . *
Byte 5 :   10001000  =  88H      * . . . * . . .
Byte 6 :   01000100  =  44H      . * . . . * . .
Byte 7 :   00100010  =  22H      . . * . . . * .
Byte 8 :   00010001  =  11H      . . . * . . . *
```

When executing a "FILL" instruction using this image, the area affected will be hatched.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

For line graphics, hatchings, etc., bit images Ø to 12 are already predefined. They may be redefined anytime, if desired:

```
 Ø = 1ØØ percent fill
 1 = 75 percent fill
 2 = 5Ø percent fill
 3 = 25 percent fill
 4 = 1Ø percent fill
 5 = Hatching from left to right
 6 = Hatching from right to left
 7 = Cross-hatching
 8 = Vertical lines
 9 = Horizontal lines
1Ø = A "D&Z" glyph (ah, well,...)
11 = Vertical serpentine lines
12 = Horizontal serpentine lines
```

D.4. A Programmimg Example

Here is a simple programming example in BASIC: Let us assume you have 1ØØ data stored in an array called Y. The program will plot the coordinate points on the screen and connect them by lines.

```
  1 DIM Y(1ØØ)                    : REM Declare array
  2 DEF FNY (Y) = 4ØØ - (4 * Y) : REM Transformation function for Y
 1Ø DIFF = 64Ø / 1ØØ              : REM Increment of X
 2Ø REM
 3Ø REM (here you would insert the code to set up the Y array)
 4Ø REM
1ØØØ REM *** Graphics Output Section ***
1Ø1Ø PRINT CHR$ (27); "I"                : REM Start Interpreter
1Ø2Ø PRINT "INIT" : PRINT "CRT 1,1,Ø,1" : REM Define screen
1Ø3Ø REM Set graphics cursor to beginning of curve:
1Ø4Ø PRINT "SET Ø,"; FNY (Y (Ø))
1Ø5Ø REM Plot output loop
1Ø6Ø I = 1 : FOR X = DIFF TO 639 STEP DIFF
1Ø7Ø         PRINT "TO " ; X ; "," ; FNY (Y (I)) : I = I + 1
1Ø8Ø         NEXT X
1Ø9Ø REM Inactivate Interpreter:
11ØØ PRINT "END"
9999 END
```

For detailed explanations of the various graphics instructions, please refer to the corresponding paragraphs.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.5. Table of Graphics Instructions

Activating Graphics: ESC I (ESCAPE, uppercase "I")

Instructions:

INIT - Initialize Graphics
CRT - Define Output Screen
BKGR - Select Background Colour
COLR - Select Plot Colour
MASK - Define Line Mask
MODE - Set Plot Mode
IMG - Define New Bit Images
CLR - Erase Graphics
SET - Set Pixel
RES - Reset Pixel
LINE - Draw Line
TO - Continue Drawing Line
RECT - Draw Rectangle
FILL - Fill Rectangular Area
CIRC - Draw Circle
ARC - Draw Arc of Circle
TEXT - Write Text
COPY - Perform Hard Copy
END - Terminate Graphics

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6. Explanation of Graphics Instructions

In the subsequent paragraphs, each graphics instruction is precisely described. A few words in advance: There is a marvellous way of testing graphics programs by just omitting the "<ESC> I" sequence at the beginning. You will then see all "graphic" instructions as ordinary text on the screen. Furthermore, it is recommended to get acquainted with the graphics instructions using the special mode of the command interpreter.

D.6.1. INIT - Initialize Graphics

Instruction: INIT

This instruction resets the entire graphics to its initial state. It should be the first instruction before any other graphics command is executed.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6.2. CRT - Define Output Screen

Instruction: CRT scrno,text,screen,colour

As your computer has two video outputs for CRT monitors, the output screen must be specified further. Four parameters have to be passed, these are:

scrno - Sreen number. The numbers 1 or 2 are permitted, all other specifications will lead to screen 1 to be used for output.

text - If you specify "Ø" here, normal text output will be suppressed, whereas a "1" causes graphics and text to be mixed on the screen.

screen - Here you specify whether a colour monitor is connected. A "Ø" means that you are using the ordinary green monitor, a "1" stands for a colour monitor.

colour - This parameter must lie within the Ø to 7 range. If specifies which partial colours are to be displayed on the screen. The values and their meanings are:

Ø - No output at all
1 - Display blue parts
2 - Display red parts
3 - Display blue and red parts
4 - Display green parts
5 - Display blue and green parts
6 - Display red and green parts
7 - Display all colours

Any other values will cause all colours to be displayed.

```
*********************************
*    Graphics Interpreter       *
*********************************
```

D.6.3. BKGR - Select Background Colour

Instruction: BKGR n

The background colour is completely independent of the plot colour. It is defined entirely by the screen hardware and does not occupy any memory. The value of n must be in the Ø to 7 interval, other values will be replaced by 7 (white).

D.6.4. COLR - Select Plot Colour

Instruction: COLR n

The plot colour determines the colour of nearly all subsequent graphics instructions. The only exception to this rule is the "FILL" instruction. If the value of n is not in the Ø to 7 range, white (7) is assumed.

D.6.5. MASK - Define Line Mask

Instruction: MASK bit pattern

Each line you draw is constructed of a number of image dots or "pixels". The graphics interpreter goes one step beyond: it composes lines from multiples of a 16-bit vector. Normally, all bits in this vector are set to "1", thus creating an uninterrupted line. But by redefining the vector, you are able to plot also dashed or dotted lines. The vector contents are specified as a chain of zeroes ond ones. If characters should miss, the vector is passed rightbound with leading zeros. If you specify a too long string of more than 16 ones and zeroes, only the last 16 characters will be regarded.

Here are two examples. To create a dashed line, you would specify a vector as follows:

```
MASK 1111000011110000
```

If you prefer dotted lines, your vector would look like:

```
MASK 1000100010001000
```

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6.6. MODE - Set Plot Mode

Instruction: MODE n

If you want, you are free to select another plot mode than the standard overwrite mode. There are four different modes at your disposal:

Ø - Overwrite. Each output overwrites old graphics. This is the normal mode.

1 - Invert colour. Plotting is done using the inverted plot colour.

2 - Erase colour. According to the colour code specified with the "CRT" instruction, partial colours are erased.

3 - Merge. The plot colour specified is added to existing graphic informations.

All other values will cause mode Ø to be selected.

D.6.7. IMG - Define New Bit Images

Instruction: IMG n,bit1,bit2,bit3,bit4,bit5,bit6,bit7,bit8

This instruction has been explained further above. "n" is a number in the Ø to 126 range, other values will be replaced by 126. This number corresponds to a character in the ASCII code list. Hence, to redefine the bit pattern of the letter "A", you would specify a value of 65 for n, as "A" is character No. 65 in the ASCII alphabet. The parameters "bit1" to "bit8" represent eight vectors of 8 bits each. A set bit is marked by a "1", a reset bit by "Ø". If bits miss, the corresponding vector is passed rightbound with leading zeroes. If there are more than 8 bits specified, only the trailing 8 bits are regarded.

For example, the definition of the "A" letter described at D.3 looks like this:

```
IMG 65,ØØ11ØØØ,ØØ1ØØ1ØØ,Ø1ØØØØ1Ø,Ø1111111Ø,Ø1ØØØØ1Ø,Ø1ØØØØ1Ø,Ø1ØØØØ1Ø
```

You see that only 7 bit vectors are specified, because the 8th one has a value of zero and could be omitted.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6.8. CLR - Erase Graphics

```
Instruction:    CLR                          or
                CLR   x1,y1,x2,y2,colour
```

This instruction erases the entire graphics, or only parts of it. If you use the instruction without any parameters:

```
CLR
```

all graphics is erased. On the other hand, a parameter list like:

```
CLR x1,y1,x2,y2,colour
```

would have the following effect: x1,y1 denote the upper left corner of a rectangle, x2,y2 the lower right one. Within the area of this rectangle, all parts are erased which correspond to the value of "colour". "Colour" may assume following values:

```
Ø - No erasure at all
1 - Erase blue parts
2 - Erase red parts
3 - Erase blue and red parts
4 - Erase green parts
5 - Erase blue and green parts
6 - Erase red and green parts
7 - Erase all colours
```

Any other values will cause all graphics to be erased.

D.6.9. SET - Set Pixel

```
Instruction:    SET x,y
```

The pixel at the coordinate point X,Y is set, i. e. lit up.

D.6.1Ø. RES - Reset Pixel

```
Instruction:    RES x,y
```

The pixel at the coordinate point X,Y is reset.

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6.11. LINE - Draw Line

Instruction: LINE x1,y1,x2,y2

A straight line is drawn from the point at coordinates x1, y1 to the point at coordinates x2, y2. The line mode can be defined by a previous call of the "MASK" instruction.

D.6.12. TO - Continue Drawing Line

Instruction: TO x,y

A straight line is drawn from current position of the (invisible) graphic cursor to the point at coordinates x,y. By means of this function, polygonials can be drawn, as the programming example shows.

D.6.13. RECT - Draw Rectangle

Instruction: RECT x1,y1,x2,y2

A rectanlge is drawn, the upper left corner of which is determined by the coordinates x1,y1, and the lower right by coordinates x2,y2.

D.6.14. FILL - Fill Rectangular Area

Instruction: FILL x1,y1,x2,y2,image,colour

x1,y1 denote the upper left, and x2,y2 the lower right corner of a rectangular area. This area is filled with a bit pattern. The parameter "image" denotes a number between 0 and 126, according to the ASCII alphabet. You may fill the area with one of the predefined fill patterns 0 to 12, with printable ASCII characters (whose values are in the 33 to 126 range), or with a pattern of your choice you have defined previously by means of the "IMG" instruction. The parameter "colour" specifies the colour to be used with the fill pattern.

For example, let us fill half of the screen with our company logo (D&Z) in red:

```
FILL 0,0,200,399,10,2
```

****
* Graphics Interpreter *
****

D.6.15. CIRC - Draw Circle

Instruction: CIRC x,y,r

A circle of radius r is drawn around the centre specified by the coordinates x, y.

D.6.16. ARC - Draw Arc of a Circle

Instruction: ARC x,y,r,phi1,phi2

A circle arc of radius r is plotted centered at coordinates x,y. The starting and final angles, phi1 and phi2, respectively, must be specified in degrees from Ø to 359. The Ø degrees direction points to the right edge of the screen, and 9Ø degrees to the upper edge. The angle is counted conterclockwise, as usual in mathematics.

Example: Plot a quarter-circle around the centre 1ØØ,1ØØ with a radius of 5Ø in the first quadrant:

```
ARC 1ØØ,1ØØ,5Ø,Ø,9Ø
```

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6.17. TEXT - Write Text

Instruction: TEXT x,y,rot,size,text

This function provides for a general facility to output texts. Size and writing direction can be chosen as required. Furthermore, italic character mode is available. It should be noted that the size specification is only course, because the actual size depends on the writing direction. The parameters are:

x,y - Start point of text. This point denotes the lower left corner of the 8 x 8 matrix of the first letter.

rot - Write direction. The possible values go from Ø to 7, where the direction (angle) is calculated from (45 * rot). Thus, a "Ø" means the usual direction, a "1" a direction to the upper right, a "2" vertically upwards, and so forth.

If you add 1Ø to the values desribed above, the text will be written in italic mode. Values from 1Ø to 17 have the same effect as Ø to 7, with the exception that you obtain italic characters.

size - Character size, which may vary from 1 to 16. The characters are magnified by the factor "size", as compared to standard screen output. A "1" means no magnification at all (standard size), a "2" effects a 1:2 scale, and so on up to a 1:16 scale, which is the largest size applicable.

The text itself is the whole remainder of the line, beginning just after the comma behind "size", including leading blanks.

Example: Write "Hey, there!" head down, beginning at coordinates 1ØØ,1ØØ, in twice the normal size:

```
TEXT 1ØØ,1ØØ,4,2,Hey, there!
```

```
*******************************
*    Graphics Interpreter     *
*******************************
```

D.6.18. COPY - Hard Copy on the Ink Jet Printer

Instruction: COPY colour

The contents of the graphic screen are transferred to the SHARP ink jet printer IO-7ØØ. You can specify which colour parts you want to be copied:

Ø - No output at all
1 - Copy blue parts
2 - Copy red parts
3 - Copy blue and red parts
4 - Copy green parts
5 - Copy blue and green parts
6 - Copy red and green parts
7 - Copy all colours

Any other values will cause all colours to be copied.

D.6.19. END - Terminate Graphics

Instruction: END

This instruction merely switches the screen back to normal text mode, if the graphics mode has been activated by the control sequence <ESC> I. Otherwise, it has no effect at all. It is, however, good customs to terminate each program using graphics by this instruction.

E O S

Operating System for
Personal Computers
with a Z80-CPU

Version 3

System Manual

Contents

C o n t e n t s

Within this manual, the logic design and the internal structure of the EOS operating system kernel is described, along with its system calls. Profound knowlegde of the Z80-CPU and its assembly language is a prerequisite to understand and successfully use the information contained herein. If improperly used, some of the system calls may cause loss of data or system breakdowns. It must be emphasized that assembly language programming including the application of system calls is definitely no job for novices.

The manual is divided into four sections. The first section contains a description of the EOS system kernel interface. System calling mechanisms and the related data structures are dicussed in detail. The second section is devoted to the interface to the MZ-3500 computer system hardware, describing its system functions and data structures. The third section refers to the graphics interface. All assembly language facilities for using graphics are listed there. In the final section, the CRT (screen) and its implemented control functions are dealt with.

Section A

EOS system interface

A.1. Preview

This section treats the user interface to the EOS operating system kernel. The interface is compatible to the operating systems CP/M and DiCOS; versions CP/M 3.Ø and DiCOS 1.1 are supported. Furthermore, disk drive management including the related data structures, and the facilities to control the system behavior by setting internal system variables are discussed. The hardware, graphics and CRT interfaces are described in sections of their own.

A.2. System call conventions

All system calls have the same convention: Register C is loaded with a function code, additional parameters (if any) are loaded into register E or into register pair DE, respectively. Results from the system are returned in register pair HL, if any. Moreover, register A is loaded with the contents of L, and register B with the contents of H, if not stated otherwise. The call itself is performed by a subroutine CALL to address 0005H. For example, if a character is to be sent to the screen, the system call would look like this:

```
LD      C,2             ; function code
LD      E,'*'           ; output an asterisk
CALL    5               ; SYSTEM CALL
```

The system maintains a stack area of its own. It is thus not necessary for the user program to supply extra stack for system calls.

A.3. I/O device assignments

With EOS, I/O device assignment has been completely redesigned. Five logical devices have been defined. These devices are:

Console Input	for functions 1, 6, 10, and 11
Console Output	for functions 2, 6, 9, and 111
Aux Device Input	for functions 3 and 7
Aux Device Output	for functions 4 and 8
List Output	for functions 5 and 112

Physical devices may be freely defined by the system implementor. A device table is constructed to describe the properties of the devices as follows:

```
DB    'NAME  '        device name, 6 Bytes
DB    ATTRIB          attribute byte
DB    BAUD            software adjustable Baud rate
```

The attribute byte is a bit vector with the following meanings:

```
Bit 0 set - Device can perform input
Bit 1 set - Device can perform output
Bit 2 set - Baud rate is software-adjustable
Bit 3 set - Device uses CTS - protocol
Bit 4 set - Device uses XON/XOFF - protocol
```

All physical I/O devices except for the disk drives are defined in this table. The devices are numbered internally, the first table entry is number 0, the second one number 1, and so forth. Each logical device has a 16-bit vector associated for up to 16 physical devices, where the most significant bit corresponds to physical device 0, the second to physical device 1, and so on. To map a physical device to a logical, simply the corresponding bit in its vector is set.

This concept allows for more than one physical device being allocated to a logical device by setting more than one bit in its mapping vector. If so, the system reaction is as follows:

When an input device is referred to, all mapped physical devices from 0 upwards are tested for a pending character. The first device ready for input is then read by EOS.

If an output device is referred to, the character is sent to all mapped output devices. EOS waits until the current device is ready to accept the character, before referring to the next device. The wait interval may be set by system function 73. Should the wait time elapse without the device becoming ready, a message is displayed on the screen:

```
Device <name> not ready
```

The construction "<name>" is replaced by the name of the non-ready physical device. Then, the input device CONIN: is checked for a pending character. If so, the character is read and the output operation aborted. If the printer is switched parallel to the output device, it is switched off, too. If there has no key been touched on the keyboard, the wait cycle will be repeated.

Physical device 0 is treated specially, as it is declarated as a CRT terminal. As a consequence, no error messages will be displayed if the CRT screen is not ready. If the printer is switched parallel to the screen due to an entry of the Ctrl-P character (see EOS functions 1 and 10) and the printer itself is assigned to the CRT terminal, no printer echo will take place to inhibit double printing on the screen.

The following devices have been implemented on the MZ-3500:

SCREEN	- the CRT screen
KEYBD	- the keyboard
RS232	- the built-in V24 / RS232 interface
CENTRO	- the built-in Centronics interface
CE332P	- the SHARP CE-332P printer
GRAPH	- the graphics interpreter
DUMMY	- a dummy device, which accepts any kind of output and always returns 1AH (Control-Z, the end-of-file character) for input.

The BAUD byte holds the following information on the RS-232-C interface:

0 =	50 baud	8 =	1800 baud
1 =	75 baud	9 =	2000 baud
2 =	110 baud	10 =	2400 baud
3 =	134 baud	11 =	3600 baud
4 =	150 baud	12 =	4800 baud
5 =	300 baud	13 =	7200 baud
6 =	600 baud	14 =	9600 baud
7 =	1200 baud	15 =	19200 baud

If bit 7 is set, the transfer format is 5 data bits and 1 1/2 stop bits, otherwise 8 data bits and one stop bit. Even parity can be specified by setting bit 6, otherwise, parity is suppressed. To alter the BAUD byte contents, use EOS function 50 (see also section C, function 21).

A.4. File control block FCB

All system calls referring to disk files expect a pointer to a file control block, which will be abbreviated as "FCB". An FCB is at least 33 bytes long, or 36 bytes for random mode, and supplies following information to the system:

Byte 1: Drive number where the file is located. If the default drive is referred to, this byte is set to Ø. Otherwise, a value of 1 means drive A, 2 drive B, and so forth up to 16 for drive P. In the directory entry, Byte 1 denotes the user area the file is associated to, or it is set to ØE5H for a non-existent or erased entry.

Bytes 2- 9: File name. If the name is shorter than 8 characters, the field has to padded with space characters. Only the least significant 7 bits of each byte are evaluated during directory searches, EOS regards the most significant bit as an attribute. The attribute bits are partially predefined:

Byte 2 (f1'): free for user purposes.
Byte 3 (f2'): free for user purposes.
Byte 4 (f3'): free for user purposes.
Byte 5 (f4'): this file will be erased after despooling resp. after execution of all commands contained in it.
Byte 6 (f5'): interface bit for system calls.
Byte 7 (f6'): interface bit for system calls.
Byte 8 (f7'): - reserved for EOS -.
Byte 9 (f8'): - reserved for EOS -.

Bytes 1Ø-12: File type. The file type is interpreted in a manner comparable to the file name, with the following meanings of the most significant bits:

Byte 1Ø (t1'): File is write protected.

Byte 11 (t2'): System attribute. This file will not be affected by built-in commands like DIR, ERA as long as this is not expressively specified. The file may be opened from a different user area when it is located in the user area Ø, but only in read only mode.

Byte 12 (f3'): Backup attribute. This bit may be set by any program (i.e. a file backup utility). On each file alteration, EOS will reset this bit.

For CP/M compatibility, the meaning of this bit has been reversed, as referred to DiCOS.

Byte 13: Extent number. The extent number begins at Ø for the first record and is incremented by 1 after each 128 records. Byte 5 holds 5 bits of the extent number, further 6 bits are stored in byte 15.

Byte 14: Within the directory, this is the byte count. When the file has been opened, byte 14 holds certain EOS flags, which must not be changed by the user.

Byte 15: Second extent byte. The lower 6 bits represent the upper 6 bits of the extent number.

Byte 16: Record number within the extent whose number is stored in bytes 13 and 15. Always less than 129. The total number of records of a file may be computed from (extent number * 128) + this byte.

Bytes 17-32: Allocation map. Here the system enters the numbers of the clusters occupied by the file. If an overflow occurs, a new directory entry is established.

Byte 33: Current block number. This byte denotes the number of the block within the current extent which is to be processed next. When a file is opened, the first 32 bytes of the directory entry are copied into the FCB. Consequently, byte 33 should be set to Ø before opening a file, as otherwise the system begins reading resp. writing at the record number found there !

Bytes 34-36: Random block number. This is a 3-byte number used for random access disk I/O. As usual on the Z-8Ø, it is stored in reverse order, i. e. least significant byte first (in byte 34), most significant byte in byte 36.

A.5. Special directory structures

A number of new data structures have been introduced to allow for storing date and time within the directory. These are directory labels and time stamps. An entry of this kind is always 4 bytes long. It is defined as described at system functions 104 and 105:

Bytes 1-2 : Julian Date, where 1 denotes January 1st, 1978.
Byte 3 : Hour, BCD format
Byte 4 : Minute, BCD format

A.5.1. The directory label

Time stamps are written under the control of a so-called "directory label". Technically, the label is nothing else but a normal directory entry in user area 32. The 11 characters of the file name and the file type may be used to assign a name to the data carrier (the disk) itself.

The directory label is constructed as follows:

Byte 1 : a value of 32 to denote the directory label.

Bytes 2-11 : Data carrier name.

Byte 12 : The "data byte", a bit vector carrying following information:

Bit 6 - Perform time stamps on file accesses.
Bit 5 - Perform time stamps on file alterations.
Bit 4 - Perform time stamps on creations of new files.

The remaining bits are reserved. Either bit 4 or bit 6 may be set in the data byte, as there is only one field for both kinds of time stamps. Time stamping on all file accesses thus excludes time stamps on creation of new files, and vice versa.

Bytes 13-24 : - reserved -

Bytes 25-28 : Moment of directory label creation.

Bytes 29-32 : Moment of last change of directory label.

A.5.2. Time Stamps

For time stamps, special directory entries are required. The system does not create them, they have to be established by means of the utility program $INITDIR. These entries then occupy the last one of the four directory entries per 128-bytes block. Such a "Time - FCB" is flagged by a value of 33 in the user byte. Beginning with the second byte, groups of 10 bytes each are reserved for each of the other three (normal files) possible entries in the 128-bytes block.

This means that the first 10 bytes (bytes 2 to 11) will recieve the time stamps of the first directory entry of the current block, the second 10 bytes (bytes 12 to 21) the time stamps of the second entry, and the last 10-byte group (bytes 22 to 31) the ones of the third directory entry. The very last byte (32) of the "Time - FCB" is reserved.

Such a 10-byte time stamp bears the following information:

Bytes 1-4 : Moment of creation or last access

Bytes 5-8 : Moment of last alteration

Byte 9 : Password mode, always 0 under EOS.

Byte 10 : - reserved -

It should be mentioned that EOS refuses to create directory labels if the data byte requests time stamps, but no "Time - FCBs" are present within the directory.

A.6. Zero page initialization

Before starting a program, EOS initializes the Zero page, i.e. the memory within the address range from 0000H to 00FFH. Additionally specified file names, if any, are deposited at predefined areas. The command line remainder is made available, too. The Zero page is initialized exactly the same way as CP/M plus would do it. The conventions are in full detail:

000H - 002H : Jump instruction to the system restart entry. A jump to this address terminates the current program and restarts the command interpreter.

003H : Formerly, the IOByte. Set to 0 by EOS.

004H : Formerly, the User/Disk byte of CP/M's CCP. EOS loads the upper 4 bits with the current user area, and the lower 4 bits with the current drive at program start.

005H - 007H : System entry jump vector. Points to the begin of EOS, resp. to the begin of the function module loaded last. A jump to this address initiates the execution of a system function. At the same time, the address of this jump instruction denotes the first byte occupied by EOS within the user memory.

008H - 03AH : - reserved for interrupt vectors -

03BH - 04FH : - reserved for EOS -

050H : Number of the drive the current program has been loaded from. A "1" corresponds to drive A:, "2" to drive B:, etc.

051H - 052H : Pointer to the begin of the password of the first file name. This pointer points into the command line remainder, which begins at address 80H. If no password has been specified for the first file name, the field is set to 0.

053H : Length of password of first file name, or 0 if no password has been specified.

054H - 055H : Pointer to the begin of the password of the second file name. This pointer also points into the command line remainder, which begins at address 80H. If no password has been specified for the second file name, the field is zero filled.

056H : Length of password of second file name, or 0 if no password has been specified.

Ø57H - Ø5BH : - reserved -

Ø5CH - Ø6BH : FCB of first file name. If a file name could be generated out of the word following the command name, this area contains the first 16 bytes of a FCB. Otherwise the area from Ø5DH to Ø67H is filled with spaces and the byte at address Ø5CH is set to Ø. The area from Ø5CH to Ø7FH may be used as FCB by the program. But before doing so, the second FCB which has been set up beginning at Ø6CH, should be saved if needed.

Ø6CH - Ø7BH : FCB of second file name. If the second word after the command word could be interpreted as a file name, the first 16 bytes of the so generated FCB are deposited here. If it was not possible to create a second FCB, address Ø6CH is set to zero and the area from Ø6DH to Ø77H filled with spaces.

Ø7CH - Ø7FH : free for usage as FCB block counter.

Ø8ØH : Begin of DMA buffer. Contains the number of characters of the succeeding command line remainder.

Ø81H - ØFFH : Command line tail. This is the rest of the command line immediately following the command name. Normally, the command line tail would begin with one or more spaces.

Before starting a program, the command interpreter sets a number of system variables to specific values. In detail:

```
Multisector count  = 1
DMA address        = 8ØH
String  delimiter  = "$"
Program end code   = Ø     (but not so if started by function 47).
Console mode       = Ø
```

The stack pointer is set to a stack area of 32 bytes, the top of which is initialized with Ø. If the current program performs a RETurn instruction, this causes a system warm start contrary to CP/M 2.2, where the RET instruction passed control to the CCP.

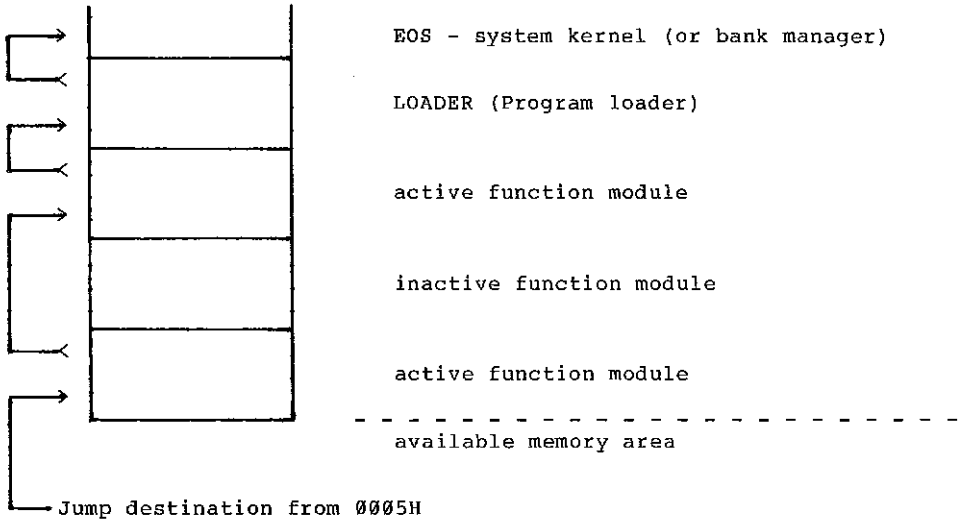
A.7. Function modules

Function modules are specific extensions of the EOS system kernel. These modules are linked by a special program to normal program files. When loading a program, this loader deposits all linked function modules directly underneath the system kernel, resp. below all other modules already loaded. Subsequently, the system entry jump vector at address 0005H is changed to point into the function module loaded last. All future system call pass through this module, enabling it to intercept or perform special tasks. Numerous programs make use of this facility, e.g. the background printing utility, the console and list I/O redirection programs, etc. The program loader itself is such a function module. Normally, it removes itself from memory after a successfull program load initiated by the command interpreter, but it can be kept in memory and its functions made available for an user program if the user program loaded is supplied with a dummy funtion module header.

A function module remains in memory until it sets a special flag signalizing that it has done its work. On each warm start, the command interpreter checks if any module has set this flag. If such a module is found, it is removed from the system call chain. If there are modules residing in memory below the dis-activated one, memory cannot be released before all "lower" modules have terminated their tasks and been removed.

A.7.1. Memory organization after loading function modules

For example, let us discuss a situation with three modules loaded into memory, one of them being inactive. The arrows show the path of system calls through the modules:



The graphic shows that system calls are passed through all function modules. Should two similar modules be loaded, both of them intercepting the same system call, the one loaded first is not referred to until the second has terminated its function. The file executor makes use of this facility to allow for nested execution of command files.

A.7.2. Program header

If a program contains function modules, it is preceded by a 256-byte header describing where the modules are located within the file. The header may additionally bear information on system variables to be changed; the loader does the appropriate assignments after loading is finished. But the header may as well be empty, in which case the loader is "tricked out" and does not remove itself from memory. It it then possible to call the loader from the program using system function 59. On the other hand, a program may consist merely of a header and the associated function modules. When such a program file is loaded, the command interpreter suppresses the search for inactive modules on its next invocation. It becomes thus possible to keep modules with their termination flag set. The subsequent program may use these functions, they are removed only after the program run being terminated.

System call 60 has been defined to communicate with modules. This call has several parameters which can be tested by all modules loaded. If none of them shows any reaction, the EOS system kernel returns 0FFH in A and 00H in H.

Let us do a view on the header. It begins with a 16-byte organization area:

```
        DB      0C9H            ; Header designation
        DW      CODESIZE        ; length of program code
        DB      0C9H,0,0        ; jump to a parameter setting routine
        DS      9               ; - reserved -
        DB      NFUNC           ; number of function modules (0-15)
```

The organization area is followed by up to 15 entries, one per function module linked:

```
        DW      OFFSET          ; offset of module in file
        DW      CODESIZE        ; Module length
        DW      EXTRA           ; RAM required additionally
        DB      'NAME    '      ; 8 bytes module name
        DW      0               ; - reserved -
```

Non-existing entries are filled with zeros.

A.7.3. Internal structure of a function module

A function module has to satisfy three conditions to be recognized by the system: It must be formatted according to the "PRL" format of Digital Research, it must be preceded by a specific header so that the program loader resp. the command interpreter can access it, and the file type must read "RSX" instead of "PRL". The easiest way to generate PRL - formatted object modules is to use Digital Research's linker, which is recommended to be purchased. The header of a PRL file contains the following information:

```
        DB      Ø               ; - reserved -
        DW      SIZE            ; size of program code
        DB      Ø               ; - reserved -
        DW      EXTRA           ; additionally required RAM
        DS      25Ø             ; - reserved -
```

Accordingly, the code starts Ø1ØØH bytes after beginning of the file. The base addres is always Ø1ØØH, so the assembler source of the module has to begin with a pseudo - instruction "ORG 1ØØH". The code is immediately followed by a bit table. For each byte within the code area, one bit flags whether the byte has to be adapted to the current loading address or not. The first bit (bit 7) of the first byte of the bit table denotes the correction for the first code byte, bit 6 the correction for the second byte, and so forth. The bit table begins at address

start-of-file + Ø1ØØH + SIZE (size of code).

The function module itself has to be begin with the follwing data structure:

```
        DB      Ø,Ø,Ø,Ø,Ø,Ø     ; - reserved -
START:  JP      MODBEG          ; Entry into the module
OUTOF:  JP      Ø               ; Exit from the module
LASTP:  DW      LAST            ; Address of previous module
REMOVE: DB      Ø               ; Inactivity flag
        DB      Ø               ; - reserved -
NAME:   DB      'NAME    '      ; Module name
        DB      Ø,Ø,Ø           ; - reserved -
```

The labels and their meanings are:

START: This is the module entry point. The program loader alters the system entry jump vector at address 0005H so that it points to this address.

OUTOF: This is the exit out of the module. If a system call has to be issued, the module must be left by a jump to this address. The program loader alters the adress of this jump instruction in such a way that it leads into the follwing module, resp. into the program loader. The vector within the loader itself points into the EOS bank manager.

LASTP: A pointer to the previous module in the chain. The module loaded last has a value of 0005H in this point.

REMOVE: If this byte is different from zero, the module will be erased from memory at the next warm start.

NAME: Module name, eight characters.

When programming modules, stack discipline must strictly be observed, because it is possible that the calling program does not supply sufficient stack area.

A.7.4. Function modules - an example

As an example, the module shown below intercepts on system call 1 (read console character), transforming each line feed character (LF) into a carriage return (CR). Furthermore, it is disactivated by control-C:

```
          ORG     100H                ; base addresse is 0100H
          DB      0,0,0,0,0,0         ; - reserved -
COMEHERE: JP      TEST                ; Module entry
EOS:      JP      0                   ; Module exit
          DW      0,0                 ; room for pointer
REMOVE:   DB      0                   ; do not remove
          DB      0                   ; - reserved -
          DB      'LF - CR '          ; Module name
          DB      0,0,0               ; - reserved -

TEST:     LD      A,C                 ; check system call,
          CP      1                   ;  is it function 1?
          JR      NZ,BDOS             ; pass if not so
          LD      (SPSAVE),SP         ; otherwise, save stack pointer
          LD      SP,STACK            ;  and set up own stack area
          CALL    EOS                 ; call EOS
          LD      SP,(SPSAVE)         ; restore old stack
          CP      0AH                 ; Line Feed?
          JR      Z,LF                ;  if so, branch
          CP      03H                 ; Control-C?
          RET     NZ                  ;  if not, return character
          LD      0FFH                ; otherwise,
          LD      (REMOVE),A          ;  set termination flag
          JP      0                   ;  and perform warm start
LF:       LD      A,0DH               ; replace LF by CR
          RET

SPSAVE:   DW      0                   ; room for old SP

          DS      4                   ; auxiliary stack area

STACK     EQU     $                   ; top of own stack

          END
```

A.8. System variables

All system kernel variables have been collected into a memory area of their own. This area is placed in the "common memory" area, on the MZ-3500, it begins at 0FF00H. The data contained there can be read and written. To access system variables, system call 49 should be used, as the base address of the variables' area is implementation dependent.

The table below has five columns, the first of which denotes the offset of a certain variable relative to the area base address, the second the variable name. Columm three specifies whether the variable has the same meaning as the corresponding CP/M plus variable. Column four specifies whether the variable may be altered by the user, or it is "read-only".

Offset	Name	CP/M	Alter	Function
00H-04H			no	- reserved -
05H	EOSVERS	yes	yes	EOS version number for function 12. Can be changed by function 72.
06H-09H	USERFLG	yes	yes	four bytes free for user applications.
0AH-0FH			no	- reserved -
10H-11H	RETCODE	yes	yes	Program return code for function 108.
12H-17H			no	- reserved -
18H-19H	FILENR	no	no	The number stored in this variable is used by $SPOOL and $DO to create unique file names for temporary files. It is incremented by 1 after each generation of a temp file. The file name generated will be "SYSnnnnn.$$$".
1AH	CWIDTH	yes	yes	Characters per screen line. Not used by EOS.
1BH			no	- reserved -
1CH	CLENGTH	yes	yes	Number of screen lines. Not used by EOS.
1DH	TABSTOP	no	yes	current Tab stop. This value can by changed by use of function 74.
1EH-1FH			no	- reserved -

20H-21H	FXDVEC	no	no	Fixed media vector. All drives denoted as installed fixed by their corresponding DPB set a bit in this vector on their first access. After a DISK RESET (see EOS function 13), the contents of this vector are copied into the login vector.
22H-23H	CIVEC	yes	yes	CONIN: device bit vector.
24H-25H	COVEC	yes	yes	CONOUT: device bit vector.
26H-27H	AIVEC	yes	yes	AUXIN: device bit vector.
28H-29H	AOVEC	yes	yes	AUXOUT: device bit vector.
2AH-2BH	LOVEC	yes	yes	LSTOUT: device bit vector.
2CH			no	- reserved -
2DH	PRTTIME	no	yes	Time-out constant for output devices. May be changed by function 73.
2EH	DOLVL	no	no	Current nesting level of $DO executions.
2FH-32H			no	- reserved -
33H-34H	CMODE	yes	yes	Console mode vector. Can be set and read by function 109.
35H-36H	BNKBUF	yes	no	Address of the EOS workspace buffer. This buffer has a length of 128 bytes and may be used by an user program as long as no system call is performed.
37H	PRTEND	yes	yes	String delimiter for function 9. This value can be read and set by function 110.
39H-3BH	SERNR	no	no	System kernel serial number.
3CH-3DH	CURDMA	yes	no	Current DMA address, can be set by function 26.
3EH	CURDSK	yes	no	Current drive, can be set by function 14.

Offset	Name	CP/M	Alter	Function
3FH-40H	PARAM	yes	no	Parameter (contents of DE) of last EOS function.
41H			no	- reserved -
42H	BRKCH	no	yes	Current second break character. Can be set by function 75.
43H	FUNC	yes	no	Number of last EOS function.
44H	CURUSR	yes	no	Current user area. Can be read and set by function 32.
45H	DSPDRV	no	yes	The drive displayed in the status line.
46H-49H			no	- reserved -
4AH	SECCNT	yes	yes	Current Multisector count, can be set by function 44.
4BH	?ERROR	yes	yes	Current error mode, can be set by function 45.
4CH-4FH	DRVCHN	yes	yes	Drive search sequence for commands, used by command interpreter. Up to 4 entries are possible, where a "1" denotes drive A, a "2" drive B, etc. A value of zero refers to the current drive. The remainder of the table is filled with zero bytes.
50H	TMPDRV	yes	yes	Drive to be used for temporary files. This variable is evaluated by the spooler and by $DO.COM, e.g. A value of "0" denotes the current drive, a "1" drive A, a "2" drive B, and so forth.
51H	ERRDRV	yes	no	Number of drive which caused the current physical error.
52H-54H			no	- reserved -
55H	GRAPH	no	no	when non-zero, the graphics interpreter is active.
56H			no	- reserved -

Offset	Name	CP/M	Alter	Function
57H	EOSFLAG	yes	no	Current EOS flags. The flag bits may be set by function 76. for an explanation of the flag bits, refer to chapter A.9.
58H-59H	DATE	yes	no	Current Julian date, where a value of 1 corresponds to January 1st, 1978. The date and time variables can be read and set by function 1Ø4 and 1Ø5, respectively.
5AH	HOURS	yes	no	Current hour, BCD format.
5BH	MINUTES	yes	no	Current minute, BCD format.
5CH	SECONDS	yes	no	Current second, BCD format.
5DH-61H			no	- reserved -
62H-63H	MXTPA	yes	no	Top of memory, actualized by the program loader.
64H-65H			no	- reserved -
66H	DIRLBL	no	no	Directory label of current drive. May be read by function 1Ø1.
67H-69H			no	- reserved -
6AH-6BH	PH$CSV	no	no	Address of checksum vectors of current drive.
6CH-6DH	PH$ALV	no	no	Address of allocation vector of current drive. May be read by function 27.
6EH-7FH	DPB	no	no	Disk Parameter Block of current drive. A call of function 31 returns this address.

After system calls 13 (Reset Disk system) and 14 (Select Drive), the contents of the area from Ø6AH to Ø7FH are undefined. Only on physical disk accesses, values are entered there. It is recommended to read these variables only by the corresponding EOS functions.

A.9. EOS Flags

EOS maintains a flag byte in its system variables area. It is described in chapter A.8 under the name "EOSFLAG" with an offset into the system variale' area of 57H. This flag byte holds a number of user - alterable flags which control the overall behavior of EOS. The flags are not affected by system restarts. Their meanings are:

Bit 7 : - reserved to maintain CP/M plus compatibility -

Bit 6 : - reserved to maintain CP/M plus compatibility -

Bit 5 : If this bit is set, the Line Editor is activated. Upon a call of EOS function 1Ø to read an input line from the console, this line will be read using the screen editor of the computer system. Normally, this kind of input is performed by the Mini editor of EOS.

Bit 4 : If this bit is set, a break character keyed in at the console will not automatically abort the program. Instead, the break condition is displayed at the screen and the operator is requested to respond.

Bit 3 : This bit has been introduced to provide for a general break facility. If it is set, EOS checks for break characters on each call. If a break character has been recognized, the address of the corresponding system call is diplayed in conjunction with a break message. The Operator is offered various options for responding.

It shall not be kept secret that there are certain disadvantages with this facility. System throughput goes down, "typing ahead" is no longer possible and some programs processing break characters as normal input won't work properly. In general, this feature is more a debugging aid.

The remaining three bits are reserved for internal use. Bits 6 and 7 may be freely used by application programs, but if done so, they are no longer compatible to CP/M. For such purposes, it is recommended to make use of the dedicated User flags within the system variables' area.

A.10. System Errors

All system calls which access disks may cause errors. These errors are subdivided into logical and physical errors. Contrary to physical errors, the reason of logical errors is erroneous behavior of the program or the system. A typical logical error were an attempt to read data past the end of a file.

Logical errors are returned in register A. Following values are possible:

1 = Missing data. This error can occur only when reading a disk file, namely, if one attempts to read past the end of the file or within a gap produced by random write operations. Furthermore, on sequential writes, the error denotes that the directory is full.

2 = Disk is full. Can occur only with write operations.

3 = An error was encountered when changing the current extent. In general, this error will occur only when the directory structure is destroyed.

4 = Attempt to position to non-existing extent. Can occur only with random read operations. Essentially, this is an extension of logical error 1.

5 = Directory full. It is not possible to establish another extent.

6 = The random block number specified exceeds the upper limit (currently, 65535). This error can also occur on an attempt to create a file larger than 8 Megabytes.

If a physical error is encountered and normal error handling is specified, an error message is displayed on the screen. This message looks like:

```
EOS system error in function #n, file d:nnnnnnnn.eee
(message)
```

or

```
EOS system error in function #n, drive d:
(message)
```

where n denotes the system function number which lead to the error, and nnnnnnnn.eee is the file affected. The error messages themselves read like this:

1 = Read/write error. A non-recoverable error was encountered on reading or writing a disk.

2 = Device is write protected. This can occur either after a precipitate disk change or as a consequence of function 28.

3 = File is write protected. This kind of error will occur on attempts to write to protected files. There are two reasons for files being write protected:

1. If the corresponding attribute bit is set.
2. If the file resides in user area Ø, but is opened from a different user area.

4 = Device not selectable. This error occurs on attempts to access a non-existing drive or a drive which' format could not be determined.

5-7 = - reserved -

8 = File already exists. Can occur with functions 22 (CREATE FILE) or 23 (RENAME FILE).

9 = File name contains "?". A number of functions require an unambiguous filename and hence, refuse ambiguous filenames by returning with this error code.

1Ø = - reserved -

11 = A BREAK character has been recognized.

Physical errors are returned in register H, with register A set to ØFFH. A value of ØFFH is, however, also possible with logical errors, in which case H is returned as "Ø". With EOS, physical errors can be handled in various ways. The error handling mode is specified by system function 45. There are three options:

1. The error condition is flagged to the calling program.

2. The operator is notified of the error condition by a message on the screen, and the calling program is also informed.

3. The operator is notified of the error condition and is offered to take certain actions. If the operator decides to continue, the error is flagged to the calling program. The operator options are:

A. Try to repeat the operation affected.
B. Abort current program.
C. Ignore error.
D. Continue and return the error to the calling program.

A.11. Table of available system calls

No	Name	Parameters	Result
0	Terminate program	--	(does not return)
1	Console input	--	A = char.
2	Console output	E = char.	--
3	Aux. device input	--	A = char.
4	Aux. device output	E = char.	--
5	Printer output	E = char.	--
6	Terminal - I/O	E = char./code	A = char., if E = code
7	Aux. input state	--	A = 0 or 1
8	Aux. output state	--	A = 0 or 1
9	String output	DE ^ string	--
10	Read input line	DE ^ line buffer	buffer filled with input
11	Get terminal state	--	A = 0 or 1
12	Version number	--	HL = 0031H
13	Disk Reset	--	AH = error code
14	Select drive	E = drive	--
15	Open file	DE ^ FCB	AH = error code
16	Close file	DE ^ FCB	AH = error code
17	Seek directory entry	DE ^ FCB	AH = error code
18	Continue dir.-seek	--	AH = error code
19	Erase file	DE ^ FCB	AH = error code
20	Read block sequential	DE ^ FCB	AH = error code
21	Write block sequt'l	DE ^ FCB	AH = error code
22	Create new file	DE ^ FCB	AH = error code
23	Rename file	DE ^ FCB	AH = error code
24	Get login vector	--	HL = bit vector
25	Get current drive	--	A = current crive
26	Set DMA address	DE = DMA adress	--
27	Get allocation vector	--	HL = address or 0FFFFH
28	Write protect drive	--	--
29	Get R/O vector	--	HL = bit vector
30	Set file attributes	DE ^ FCB	AH = error code
31	get disk parameters	--	HL ^ DPB or = 0FFFFH
32	get/set user code	E = user / 0FFH	A = user, if E = 0FFH
33	Read random	DE ^ random FCB	AH = error code
34	Write random	DE ^ random FCB	AH = error code
35	Compute file size	DE ^ random FCB	R0-R2 hold block number
36	Set random block #	DE ^ random FCB	R0-R2 hold block number
37	Reset drives	DE = bit vector	A = 0
38	Lock drives	DE = bit vector	--
39	Release drives	DE = bit vector	--
40	Write random w/ zero	DE ^ random FCB	AH = error code

No	Name	Parameters	Result
41	Write and verify	DE ^ FCB	AH = Ø
42	Lock block	DE ^ FCB	AH = Ø
43	Release block	DE ^ FCB	AH = Ø
44	Set Multisector Count	E = # of blocks	--
45	Set error mode	E = Ø/ØFEH/ØFFH	--
46	Get disk space	E = drive	DMA buffer=free space
47	Chain program	E = chain flag	(does not return)
48	Flush buffers	--	AH = error code
49	System variables	DE ^ vector	HL = value
5Ø	Direct BIOS call	DE ^ parameter	HL = return value
59	Load program	DE ^ FCB	AH = error code
6Ø	Call function module	DE ^ parameter	HL = error code
64	Set error mode	E = Ø or ØFFH	A = previous code
65	Change process	E = process #	--
66	Memory dump	DE ^ dump vector	--
67	Test for break char.	E = char. /ØFFH	A = Ø or ØFFH
68	Execute file	DE ^ FCB	AH = error code
69	Background print pile	DE ^ FCB	AH = error code
7Ø	Printing pontrol	E = contol code	A = Ø or ØFFH
71	Check drive ready	E = drive	HL = ready code
72	Set system version	E = version #	A = version number
73	Set time out	E = time interval	A = time out
74	Set tab stop	E = tab stop	A = tab stop
75	Define BREAK char.	E = BREAK char./Ø	A = BREAK char.
76	Set EOS flags	E = flag byte	A = flag byte
98	Release data buffers	--	AH = Ø
99	Truncate file	DE ^ FCB	AH = error code
1ØØ	Write DIR label	DE ^ dir label	AH = error code
1Ø1	DIR label data byte	E = drive	AH = error code / data
1Ø2	Read time stamps	DE ^ FCB	AH = error code
1Ø3	Write XFCB	DE ^ XFCB	A = ØFFH
1Ø4	Set date	DE ^ date	--
1Ø5	Get date	DE ^ date	A = seconds
1Ø6	Set password	DE ^ password	--
1Ø7	Get serial number	DE ^ destination	destination filled
1Ø8	Set program code	DE = code / ØFFFFH	HL=code, if DE=ØFFFFH
1Ø9	Console mode	DE = mode / ØFFFFH	HL=mode, if DE=ØFFFFH
11Ø	S/R string delimiter	E = char / ØFFFFH	A =char, if DE=ØFFFFH
111	Display block	DE ^ block param's	--
112	Print block	DE ^ block param's	--
115	Invoke graphics	DE ^ parameter	HL = error code
152	Construct FCB	DE ^ parameter	HL = end-of-string / Ø

- This page has been left blank for Your notes -

```
**************************************************************************
*                                                                        *
*                    Ø - Terminate Program Run                           *
*                                                                        *
**************************************************************************
```

On entry: C = ØØH
DE = ---

On return: (does not return)

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

The calling program is terminated, control is passed to the warm start vector at address ØØØØH. This system call has the same effect as a direct jump to ØØØH.

If the calling program has set a program error code (see function 1Ø8), this will be handled by the command interpreter.

1 - Read Console Character

On entry: C = ØlH
DE = ---

On return: A = character read from console

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

The control characters Ctrl-S, Ctrl-Q and Ctrl-P have been recognized during console output only.

The next pending character is read in from the keyboard. Printable characters and the ASCII control characters CR, LF and BS are echoed back to the console device. If a HT (tabulation character) is received, the input position will be moved to the predefined tab stop. However, should raw screen display mode be selected by function 1Ø9, no tabulation will take place.

Console output may be interrupted by typing the ASCII control character DC3 (Ctrl-S), as long as this facility has not been suppressed by system call 1Ø9. In this case, DC3 is passed to the calling program.

If console output has been suspended by typing DC3 (Ctrl-S), following characters may be keyed in:

DC1 (Ctrl-Q)
or
DC3 (Ctrl-S) - Resumption of console output.

DLE (Ctrl-P) - The printer is switched on/off parallel to the screen. DLE acts like an on/off toggle switch. If raw screen display has been selected by function 109, it is not possible to control this kind of hardcopy protocol.

BREAK character - The program is aborted. Break character recognition may also be influenced by function 109.

****
*
* 2 - Write Character to Console
*
****

On entry: C = Ø2H
E = Character to be written

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

The character passed in register E is sent to the console. If the printer is switched parallel, the character is sent to the printer, too. The screen control facilities mentioned at function 1 are also available with this function.

****
*
* 3 - Read Auxiliary Device
*
****

On entry: C = Ø3H
DE = ---

On return: A = character read from Auxiliary device

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

The Auxiliary device was formerly subdivided into the devices RDR: (Reader) and PUN: (Punch).

A character is read in from the Auxiliary device. Normally, the Auxiliary device is mapped to the serial interface, but this definition may be changed freely.

****
*
* 4 - Write to Auxiliary device *
*
****

On entry: C = Ø4H
E = character to be written

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

The Auxiliary device was formerly subdivided into the devices RDR: (Reader) and PUN: (Punch).

The character passed in register E is written to the Auxiliary device. Normally, the Auxiliary device is mapped to the serial interface, but this definition may be changed freely.

****
*
* 5 - Print Character
*
****

On entry: C = Ø5H
E = character to be printed

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

The character passed in register E is sent to the printer. If the printer is not ready to accept a character, a message will be displayed after a timeout interval which may be set by system function 73. If a key is pressed now, printer output is aborted. If the printer has been switched parallel to the screen, this mode will be canceled.

6 - Direct Terminal I/O

On entry: C = Ø6H
E = output control

On return: A = State or character, if requested for

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

The parameter E = ØFEH and E = ØFDH have been introduced.

This function allows for direct control of the console device. A control code is to be passed in register E, which may take one of following values:

ØFFH - Check keyboard for a pending character. If no character has been keyed in, a value of Ø is returned in register A; otherwise A holds the character typed in.

ØFEH - Check keyboard state. If there is a character ready for input, register A is set to ØFFH, if there is no character, A is set to Ø.

ØFDH - Read character from keyboard. The system waits until a character is typed, then returns it in register A.

others - All other values are interpreted as characters to be written and sent to the console.

As this system call is a pure I/O function, none of the control features mentioned at function 1 are available.

****
*
* 7 - Check Input Status of Auxiliary device *
*
****

On entry: C = 07H
DE = ---

On return: A = State

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

Up to now, this was the function to read the IOByte at address 0003H. Because the device assignment mechanisn has been substantially changed under EOS 3.0, the IOByte is no longer supported and the function became free for other purposes.

The Auxiliary device is checked if it has a character ready for input. If so, A is set to 1, otherwise to 0.

Normally, the Auxiliary device is defined as serial interface. The definition, however, may be changed arbitralily.

```
*************************************************************************
*                                                                       *
*                  8 - Check Output status ot Auxiliary Device          *
*                                                                       *
*************************************************************************
```

On entry: C = 08H
DE = ---

On return: A = state

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

Up to now, this was the function to set the IOByte at address 0003H. Because the device assignment mechanisn has been substantially changed under EOS 3.0, the IOByte is no longer supported and the function became free for other purposes.

The Auxiliary device is checked if it is ready to accept a character for output. If so, A is set to 1, otherwise to 0.

Normally, the Auxiliary device is defined as serial interface. The definition, however, may be changed arbitralily.

9 - Print String

On entry: C = 09H
DE = pointer to string to be printed

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

By means of system function 110, the string delimiter may be defined arbitrarily.

DE points to the first character of a text string in memory. The string is sent to the console. The control facilities mentioned at function 1 are available, if they have not been suppressed by function 109. A "$" character denotes the end of the string, but function 110 allows for any character to be defined as string delimiter.

****
*
* 10 - Read Line from Console
*
****

On entry: C = ØAH
DE = Addres of bufer or ØØØØH

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

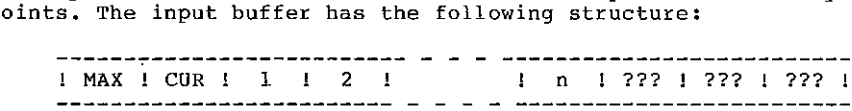
Differences to CP/M plus:

The system implementor may link a line editor to the system, e.g. to make use of special hardware features of the screen. This editor, then, can be activated resp. disactivated by function 76.

Differences to CP/M 2.2 resp. DiCOS:

The input line may be preset with characters.

An input line is read from the console and deposited in memory where DE points. The input buffer has the following structure:



MAX must be set prior to the system call and denotes the highest number of characters which can be keyed in. Input is aborted if the operator attempts to type more characters. The buffer may be up to 127 characters long.

When input is terminated, CUR holds the number of characters actually typed in. In the above scheme, the characters are numbered as 1, 2, up to n. After the last character input, the buffer contents are undefined, which is marked above as "???".

The following special characters are allowed in the input:

BREAK char. - If the very first character entered is a BREAK character, the program will be aborted. A previous call of function 109 may prevent this.

CAN (Ctrl-X) - All input is erased.

BS (Ctrl-H)
DEL (Rubout) - The character typed last is erased.

DLE (Ctrl-P) - The parallel running printer is switched on resp. off as long as this feature has not been suppressed by function 109.

Furthermore, it is possible to make the input processing implementation dependant by system function 76. By doing so, the line editing functions of CRT terminals can be made available, for instance.

If DE is set to zero, the current DMA buffer (see function 26) is used for input. The buffer may be preset with any desired text, terminated with a 0-byte. The maximum buffer size must also be specified. If the text contains a line feed (LF) or carrige return (CR) character, the function will be aborted and the string output so far be used as input. The same will occur if the text is longer than the specified maximum buffer size.

****
*
* 11 - Check Keyboard Status *
*
****

On entry: C = ØBH
DE = ---

On return: A = State

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

The keyboard status check can be limited to a test for BREAK characters.

If a character has been typed on the keyboard, a value of Ø1H is returned in A, or Ø otherwise.

By calling function 1Ø9, this function can be modified in such a way that Ø1H is returned only in case of valid BREAK characters.

****
*
* 12 - Get System Version Number
*
****

On entry: C = ØCH
DE = ---

On return: HL = Version number

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

The version number can be set to any value by function 72.

Differences to CP/M 2.2 resp. DiCOS:

see above

In register H, zero is returned (MP/M II returns 1 in H). Register L is set to 31H, denoting the compatible CP/M version number.

This function can be used to write version number dependant programs, which do not make use of the CP/M 3.Ø compatible functions when running under CP/M 2.2 or similar operating systems.

****
*
* 13 - Reset Disk System
*
****

On entry: C = ØDH
DE = ---

On return: A = Ø or ØFFH
H = physical error code

Possible logical errors:

- none -

Possible physical errors:

1 - Read/write error
4 - Drive cannot be selected

Differences to CP/M plus:

Physical error can occur.

Differences to CP/M 2.2 resp. DiCOS:

see above

All logged-in drives will be "forgotten", i.e. their definitions are erased. On the next access to a drive, all parameters are new defined, e.g. disk format. Drive A is specified as default drive. The DMA address is set to 8ØH.

Drives which are defined as being installed fixed (refer to the DPB descripition in section B) are not affected by this function, i.e. they remain defined. If it should become necessary to reset these drives too, use system function 37, which resets all drives regardless of their characteristics.

If there are yet unwritten data buffers, these are flushed to the disks. Physical errors occurring during writing will be handled according to the specified error handling mode.

14 - Select Drive

On entry: C = ØEH
E = drive to be selected for future I/O

On return: A = ØØH

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

The physical drive selection is postponed until a physical disk access.

Differences to CP/M 2.2 resp. DiCOS:

see above

The drive specified by register E is defined as reference drive for future disk I/O. The drive is not yet selected physically, this occurs on the first "real" access of the drive.

****
*
* 15 - Open File
*
****

On entry: C = ØFH
DE = address of FCB

On return: A = error code
H = physical error code

Possible logical errors:

- none -

Possible physical errors:

Ø - no errors occurred
1 - Read/write error
4 - Drive not selectable
9 - Question marks in file name

Differences to CP/M plus:

1) A password will be ignored.

Differences to CP/M 2.2 resp. DiCOS:

1) Automatic search in user area Ø und opening for reading if present there.
2) The file name must be unambiguous.
3) A byte count will be taken over.

The file specified by DE is opened. Opening a file is essentially a directory search (on the selected drive) for the file name, followed by copying the information found there into the FCB. The high extent byte is always set to zero. Normally, the lower extent byte is also set to zero, however, it is possible to specify an extent number in order to open a specific extent. The file name must be unambiguous, i.e. it must not contain question marks.

The Current Record Byte within the FCB (byte number 32) should be set to zero to enable subsequent sequential access. If it is set to ØFFH, the byte count (see function 31) is copied into this byte.

The file search is started with in the local user area. If the file is not present there, a second attempt is performed in user area Ø. Could the file be found there, the system attribute (bit t2') is tested for being set. If so, the file is opened in read only mode. This file state is flagged by setting attribute bit f8'.

This mechanism allows for accessing files in user area Ø from any other user area. It becomes thus unnecessary to have a copy of each program in each user area.

If extent Ø has been opened and time stamps on file access is specified in the directory label, time stamping is performed now.

On a successful search, A and H are both set to Ø. If the file could not be found, A is set to ØFFH and H to Ø.

In case of physical errors, a message is displayed and the program aborted in the normal error handling mode. If the error mode has been changed by function 45, A is set to ØFFH and H to one of the above mentioned error numbers.

****
*
* 16 - Close File
*
****

On entry: C = 10H
DE = Address of FCB

On return: A = error code
H = physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

A partial close is performed as normal close.
Setting of time stamps for file alterations has been transferred here from function 21.

Differences to CP/M 2.2 resp. DiCOS:

The file name must be unambiguous.

The file denoted by the FCB (pointed to by DE) is closed. If the FCB has been altered due to disk write operations, it is updated within the directory. A partial close of the file, flagged by setting file attribute bit f5', is performed as a normal close.

If the directory label specifies for time stamps on file alterations and the file has actually been altered, the current time is entered into the corresponding field in the directory entry. If the extent number is different from zero, the directory entry for extent 0 is searched beforehand.

If a physical error was encountered on directory access, an error message will be displayed and the operation may be attempted repeatedly in normal error mode. If the error mode has been changed by system function 45, register A is set to 0FFH and register H contains one of the above mentioned physical error codes.

****
*
* 17 - Search for File
*
****

On entry: C = 11H
DE = Address of FCB

On return: A = Directory code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

Function 17 allows for searching of specific files. Drive byte, file name and the first extent byte must be specified, the second extent byte is automatically set to zero. Question marks are allowed at all positions.

This function, however, will return the first matching entry found. Subsequent entries have to be searched for using system function 18. Between the "Search" calls, no other calls which cause directory accesses are allowed.

If the Drive byte contains a question mark, all directory entries of the current drive are returned, regardless whether they are allocated, free or contain time stamps. Thus, this kind of system call allows for scanning the whole directory.

Normally, a specific extent to be searched for will be specified by setting the Extent byte to the desired value. It should be noted that this operation mode can find extents up to number 31 only. However, if the extent byte is filled with a question mark, all extents of the specified files will be tranferred.

Question marks within the file name mean that the corresponding characters are irrelevant for the search, they will match any character ("wild cards"). File attributes in the "Search FCB" are ignored.

An entry found is not returned in the FCB, moreover, the complete 128-bytes dirctory block is copied into the current DMA buffer. Register A contains the number of the entry within this buffer, it can assume values from Ø to 3. If time stamps are specified on the drive searched, the values returned in A will be Ø to 2, as in this case, the last entry contains the "Time FCB".

If, for any reason, the search was unsuccessful, register A is set to ØFFH. H contains Ø if the file could not be found, otherwise the physical error code. Physical error codes are flagged only if the error mode was not changed by function 45.

****
*
* 18 - Search for subsequent entries
*
****

On entry: C = 12H
DE = ---

On return: A = Directory code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

If several directory entries are to be searched for, the directory scan process has to be initiated by function 17, and continued by using function 18. This means that between calls of function 17 and 18, no system calls may be performed which lead to eventual directory accesses. Furthermore, the FCB used by function 17 must not be altered by the user program in any way.

19 - Erase file

On entry: C = 13H
DE = Address of FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable

Differences to CP/M plus:

Passwords and erasure of XFCBs are not supported.

Differences to CP/M 2.2 resp. DiCOS:

If any of the files to be erased is write protected, no files at all will be erased.

All files matching the file name specified in the FCB will be erased. The file name may contain question marks. If file attribute f5' is set, only XFCBs are to be erased. Because EOS does not support XFCBs, this operation mode is treated as a successful erasure, i.e., actually nothing will be erased, but A and H are returned with zero values.

If one or more of the files to be erased is write protected, the erasure function will be aborted. No files at all will be erased, whether they are write protected or not.

If no file to be erased could be found, A is set to ØFFH upon return. Physical errors are treated according to the specified error handling mode.

```
**************************************************************************
*                                                                        *
*                    2Ø - Read Sequential                                *
*                                                                        *
**************************************************************************
```

On entry: C = 14H
DE = Address of FCB

On return: A = Error code
H = Number of blocks transferred, resp. physical error code.

Possible logical errors:

1 = Read past logical end of file
9 = Invalid FCB

Possible physical errors:

1 = Read/write error
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

The multisector count has been extended to 256, instead of 128.

Differences to CP/M 2.2 resp. DiCOS:

Up to 256 blocks, corresponding to 32 KBytes of data, may be transferred by one read operation.

According to the current multisector count, 1 to 256 blocks of 128 bytes each are read sequentially from disk and deposited in the memory buffer specified by function 26. Reading begins in the current extent at a position specified by the Current record byte (byte 33 in the FCB). Hence, a user program which performs sequential disk input should set the Current record byte to Ø after opening the file. When the value of the Current record byte exceeds 127, the next extent is opened automatically.

After a successful read, A and H are set to Ø. If reading had to be aborted due to an end-of-file condition or a gap in a Random file, A is set to 1 and H holds the number of blocks transferred so far, which can be Ø to 255.

Error code 9 will be returned when the file had been closed and an error occurred during closing, so that the FCB had to be marked as invalid.

If the error handing mode is set to automatic return, any physical errors are returned in register H, in which case A is set to ØFFH.

****
*
* 21 - Write sequential
*
****

On entry: C = 15H
DE = Address of FCB

On return: A = Error code
H = Number of blocks, resp. physical error code.

Possible logical errors:

2 = Disk storage exhausted
5 = Directory full
7 = Maximum file size exceeded
9 = Invalid FCB

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

1) The Multisector count has been extended to 256 max.
2) In case of errors, the file is closed automatically, so no data can be lost.
3) If time stamps upon file alterations are specified, function 16 instead of this function will perform the stamping task.

Differences to CP/M 2.2 resp. DiCOS:

Up to 256 blocks, correspoding to 32 KBytes, may be transferred by one write operation.

According to the current Multisector count, 1 to 256 blocks of 128 bytes each are written from the buffer specified by the current DMA address. The file must be opened previously. The current extent number and the Current Record byte (byte 33 in FCB) determine the block number where writing begins. A user program which has to do sequential disk output, should set the Current Record byte to Ø after opening or creating the file.

After the write operation is completed, the extent bytes and the Current Record byte denote the block to be written to next. If writing was successful, A and H are set to Ø. If a logical error occurred and writing had to be aborted, A holds the error code and H the number of blocks transferred so far; the value lies between Ø and 255.

In case of error, EOS automatically tries to close the file. Contrary to CP/M plus, no data can be lost.

If the error handing mode is set to automatic return, any physical errors are returned in register H, in which case A is set to ØFFH.

22 - Create New File

On entry: C = 16H
DE = Address of FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

5 = Directory full

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
8 = File already exists
9 = Question marks in file name

Differences to CP/M plus:

The file attribute f6' (create XFCB also) is ignored.

Differences to CP/M 2.2 resp. DiCOS:

It is no longer possible to create duplicate entries.

The file denoted by the FCB is created. Filename and Extent byte must be initalized by the calling program, no question marks are allowed in the file name. EOS initializes the FCB and opens the file for read/write accesses. The upper extent byte is always set to Ø by EOS, hence, extents with numbers between Ø and 31 may be created directly. However, it is better to avoid this, because EOS can exhibit sensitive reactions to false directory contents.

EOS ignores the file attribute f6' (create XFCB), as it does not support passwords.

If time stamps on file creation are requested, the current time is stored in the access time field on creation of Extent Ø. If time stamps on file alterations are also requested, the time is stored in the alteration time field, too.

Prior to creating the file, EOS checks if a file of the same name exists on the same disk and in the same user area. If so, the file creation request will be rejected.

Physical errors are treated according to the specified error handling mode. In case of return, A is set to ØFFH and H holds the physical error code.

23 - Rename File

On entry: C = 17H
DE = Address of FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
8 = File already exists
9 = Question marks in file name

Differences to CP/M plus:

1) Passwords are ignored.
2) The attributes of the old file name are preserved.

Differences to CP/M 2.2 resp. DiCOS:

1) Both file names must be unambiguous.
2) No file having the new name must exist on disk.

The file denoted by the FCB is renamed. In this case, the FCB contains two file names; the second one beginning at byte 17 in the FCB. Its structure corresponds to the first file name in bytes 1 to 16. The only difference is that in the second file name, the Drive byte is not evaluated. Both file names must be unambiguous, i.e. they must not contain question marks. If a file of the same name already exists on the drive, renaming will be refused by EOS. File attributes of the old file name are preserved for the new file name.

After a successful operation, both A and H are set to Ø. If the file to be renamed could not be found, A contains ØFFH and H is Ø.

Physical errors are treated according to the specified error handling mode. In case of return, A is set to ØFFH and H holds the physical error code.

24 - Get Drive Login Vector

On entry: C = 18H
DE = ---

On return: HL = Login vector

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

A 16-bit vector is returned in HL denoting all drives currently known to the system, i.e. all drives which have been accessed after the last disk system reset. The least significant bit (bit Ø in register L) corresponds to drive A, the most significant one (bit 7 in H) to drive P.

25 - Get Current Drive

On entry: C = 19H
DE = ---

On return: A = Drive number

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

The number of the drive selected by function 13 or 14 is returned in register A, where a value of Ø denotes drive A, 1 drive B, and so forth.

****
*
* 26 - Set DMA Address *
*
****

On entry: C = 1AH
DE = DMA address

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Passwords specified in the DMA buffer are ignored.

Differences to CP/M 2.2 resp. DiCOS:

- none -

The address specified by register pair DE denotes the begin of the memory buffer for all future disk I/O. The DMA buffer is used for buffering information with a number of other functions, too. Contrary to CP/M plus, passwords are ignored, as EOS does not support passwords.

Beyond disk I/O functions, the following functions make use of the DMA buffer for information transfer:

10 - read line from console
17 - search for file
18 - search for subsequent entries
46 - get free disk space
47 - chain program

****
*
* 27 - Get Allocation Vector
*
****

On entry: C = 1BH
DE = ---

On return: HL = Address of allocation vector or ØFFFFH

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Physical errors can occur.

Differences to CP/M 2.2 resp. DiCOS:

- none -

This function returns the beginning address of the allocation vector of the current drive. The "allocation vector" is a bit vector, where beginning with the most significant bit of the first byte, each bit denotes an allocation cluster. If the bit is set, the corresponding cluster on disk is occupied.

The allocation vector should not be used to determine the space remaining on a drive, as it can be placed in a different memory bank. For this purpose, function 46 should be called, which is used by all auxiliary programs of EOS.

If no current drive has yet been defined, the definitons are read. In case of physical errors during execution of this function, HL will be set to ØFFFFH on return.

****
*
* 28 - Write Protect Drive
*
****

On entry: C = 1CH
DE = ---

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

A call of this function marks the current drive as write protected. The protection can only be deleted by functions 13 or 37.

29 - Get Write Protection Vector

On entry: C = 1DH
DE = ---

On return: HL = Write Protection Vector

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Physical errors can occur.

Differences to CP/M 2.2 resp. DiCOS:

- none -

A 16-bit vector is returned in HL denoting all drives currently write protected. The least significant bit (bit Ø in register L) corresponds to drive A, the most significant one (bit 7 in H) tp drive P.

****
*
* 30 - Set File Attributes *
*
****

On entry: C = 1EH
DE = Address of FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

Passwords are ignored.

Differences to CP/M 2.2 resp. DiCOS:

1) The Byte Count has newly been introduced.
2) The Backup attribute bit has been inverted as defined in DiCOS.

The file attributes specified in the FCB are written to the directory. The attribute bits are defined as follows:

f1' - f3' = freely definable
f4' - f8' = reserved for EOS
t1' = Write protection
t2' = "System" file
t3' = File is backed-up

If attribute bit f6' is set, the Current Record field of the FCB is stored as Byte Count. This Byte Count can be read when opening the file (see also function 15).

If the attributes could successfully be set as specified, both A and H are set to Ø upon return. If the file was not found, A is set to ØFFH and H to Ø.

Physical errors are treated according to the specified error handling mode. In case of return, A is set to ØFFH and H holds the physical error code.

****
*
* 31 - Get Disk Parameter Block *
*
****

On entry: C = 1FH
DE = ---

On return: HL = Address of disk parameter block or ØFFFFH

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Physical errors can occur.

Differences to CP/M 2.2 resp. DiCOS:

- none -

The beginning address of the disk parameter block of the current drive is returned in HL.

If no current drive has yet been defined, the definitons are read. In case of physical errors during reading, HL will be set to ØFFFFH on return.

32 - Get / Set User Area

On entry: C = 20H
E = User Area or 0FFH

On return: A = User Area

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

User areas from 0 to 33 can be set.

Differences to CP/M 2.2 resp. DiCOS:

The DiCOS range 0 - 99 has been reduced.

This function returns or sets the current user area. If register E is set to 0FFH, no new user area will be set, but the current value is returned in A. If the value in E is less than 34, the current user area will be set to that value.

The range of possible values is not limited to 0 to 15 to provide for a facility to handle XFCBs in the range 16 to 31, if users want so. Furthermore, Directory labels (user area 32) and time stamps (user area 33) can be accessed directly.

****
*
* 33 - Read Random
*
****

On entry: C = 21H
DE = Address of FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

1 = Read past logical end of file
3 = Logical error in directory
4 = Random record number points to non-existing part of file
6 = Invalid Random record number

Possible physical errors:

1 = Read/write error
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

The highest possible block number is 65535 = ØFFFFH.

Differences to CP/M 2.2 resp. DiCOS:

- none -

This function is similar to function 2Ø (Read Sequential). The main difference is that the start position for disk access is denoted by a three-byte block number, which is stored in the FCB beginning at byte 34 (just after the Current record byte). The block number may assume values from Ø to 65535. It is stored in reverse order, i.e. the least significant byte is to be found in FCB byte 34, the next one in byte 35, and the most significant in byte 36. The file must have been opened prior to issuing this call.

According to the current Multisector Count, 128 up to 32768 bytes can be transferred by a single read operation. The specified block number is not affected.

It is possible to read a specific block in random mode, then switch to sequential reading for further processing. However, the system will re-read the current block on the first sequential access, resp. the first blocks if the Multisector Count is greater than 1.

In case of logical errors, register A holds an error number and H the number of successfully read blocks.

Physical errors are treated according to the specified error handling mode. In case of return, A is set to ØFFH and H holds the physical error code.

****
*
* 34 - Write Random
*
****

On entry: C = 22H
DE = Address fo FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

2 = Disk storage exhausted
3 = Logical error in directory
5 = Directory full
6 = Invalid Random record number
7 = Maximum file size exceeded

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

1) In case of error, the file will be closed automatically.
2) The highest possible block number is 65535 = ØFFFFH.

Differences to CP/M 2.2 resp. DiCOS:

- none -

This function is similar to function 21 (Write Sequential). The main difference is that the start position for disk access is denoted by a three-byte block number, which is stored in the FCB beginning at byte 34 (just after the Current record byte). The block number may assume values from Ø to 65535. It is stored in reverse order, i.e. the least significant byte is to be fuond in FCB byte 34, the next one in byte 35, and the most significant in byte 36. The file must have been opened prior to issuing this call.

According to the current Multisector Count, 128 up to 32768 bytes can be transferred by a single write operation. The specified block number is not affected.

It is possible to write a specific block in random mode, then switch to sequential writing for further processing. However, the system will re-write the current block on the first sequential access, resp. the first blocks if the Multisector Count is greater than 1.

In case of logical errors, register A holds an error number and H the number of successfully read blocks.

Physical errors are treated according to the specified error handling mode. In case of return, A is set to ØFFH and H holds the physical error code.

****
*
* 35 - Compute File Size
*
****

On entry: C = 23H
DE = Address of FCB

On return: A = Error Code
H = Physical Error Code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

The highest possible block number is 65535 = ØFFFFH.

Differences to CP/M 2.2 resp. DiCOS:

- none -

The end of the file denoted by the random-mode FCB is searched for, and the number of the next sequential block is returned in bytes 34 to 36 in the FCB. For example, if the last block occupied by the file has number 65535, 65536 will be returned even when the file consists of this single block only. It is not necessary to open the file prior to this function. If the file has been opened for writing, it must be closed, so that the directory information is updated.

If the file was found, both A and H are set to Ø upon return; if not so, A is set to ØFFH.

Physical errors are treated according to the specified error handling mode. In case of return, A is set to ØFFH and H holds the physical error code.

****
*
* 36 - Set Random Record
*
****

On entry: C = 24H
DE = Address of FCB

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

The highest possible block number is 65535 = ØFFFFH.

Differences to CP/M 2.2 resp. DiCOS:

- none -

A call to this function computes the random record number for the next block of a file processed sequentially, which is then deposited in FCB bytes 34 to 36. It becomes thus possible to switch over from random to sequential mode and back again arbitrarily.

****
*
* 37 - Reset Drive
*
****

On entry: C = 25H
DE = Bit vector

On return: A = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

- none -

The contents of DE are regarded as a bit vector, where each bit corresponds to a disk drive. The least significant bit (bit Ø in register L) denotes drive A, the next one drive B, and so forth up to the most significant bit (7 in register H), which stands for drive P.

If a bit is set, the tables of the corresponding drive is reset. All definitions, e.g. recording format, data carrier capacity etc., are redetermined on the next access of the drive. An eventual write protection of the drive is deleted.

****
*
* 38 - Lock Drive
*
****

On entry: C = 26H
DE = Bit vector

On return: A = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This is a specific MP/M II function, which is not very meaningful in a single-user system like EOS. It has been defined for compatibility purposes only. A call of this function will return the register settings as stated above.

****
*
* 39 - Unlock Drive
*
****

On entry: C = 27H
DE = Bit vector

On return: A = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This is a specific MP/M II function, which is not very meaningful in a single-user system like EOS. It has been defined for compatibility purposes only. A call of this function will return the register settings as stated above.

```
**************************************************************************
*                                                                        *
*                  40 - Write Random with Zero fill                      *
*                                                                        *
**************************************************************************
```

On entry: C = 28H
DE = Address fo FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

2 = Disk storage exhausted
3 = Logical error in directory
5 = Directory full
6 = Invalid Random record number
7 = Maximum file size exceeded
9 = Invalid FCB

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
9 = Question marks in file name

Differences to CP/M plus:

The highest possible block number is 65535 = ØFFFFH.

Differences to CP/M 2.2 resp. DiCOS:

- none -

This function is equivalent to function 34, write random. It differs in the respect that previously unallocated disk areas are filled with zero bytes, resulting in well-defined disk contents. Function 34 does not initialize new allocated disk space, which causes uninitialized areas or gaps in the file.

****
*
* 41 - Write Random and Verify
*
****

On entry: C = 29H
DE = Address of FCB

On return: A = ØFFH
H = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This is a specific MP/M II function, which is not very meaningful in a single-user system like EOS. It has been defined for compatibility purposes only. A call of this function will return the register settings as stated above.

****
*
* 42 - Lock Record
*
****

On entry: C = 2AH
DE = Address of FCB

On return: A = Ø
H = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This is a specific MP/M II function, which is not very meaningful in a single-user system like EOS. It has been defined for compatibility purposes only. A call of this function will return the register settings as stated above.

****
*
* 43 - Unlock Record
*
****

On entry: C = 2BH
DE = Address of FCB

On return: A = Ø
H = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This is a specific MP/M II function, which is not very meaningful in a single-user system like EOS. It has been defined for compatibility purposes only. A call of this function will return the register settings as stated above.

****
*
* 44 - Set Multisector Count
*
****

On entry: C = 2CH
E = Number of blocks

On return: A = Previous Multisector Count

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

The old multisector count is returned.

Differences to CP/M 2.2 resp. DiCOS:

New function.

Follwing functions are affected by the multisector count:

20 - read sequential
21 - write sequential
33 - read random
34 - write random
40 - write random with zero fill

Normally, one 128-byte block is transferred by above functions. The number of blocks read/written per system call, however, can be set to any value from 1 to 256 (where a value of zero stands for the transfer of 256 blocks). Hence, up to 32 Kbytes of data can be processed by a single disk I/O operation.

If a locical error occurs, its number is returned in register A, in which case H holds the number of successfully transferred blocks. This number is in the range 0 to 255.

Physical errors are treated according to the current error handling mode. The error code is returned in register H.

****
*
* 45 - Set Error Handling Mode *
*
****

On entry: C = 2DH
DE = Error Handling Mode

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The value passed in register E determines the future system reaction to physical errors. E may assume one of the following values:

ØFFH = No error messages will appear on the console. A physical error code is returned to the calling program.

ØFEH = An error message will be displayed, and the system returns to the calling program with the error number in register H.

any other = An error message will be displayed, and the operator can respond interactively to the situation.

46 - Return Remaining Disk Space

On entry: C = 2EH
E = Drive

On return: A = Error code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable

Differences to CP/M plus:

The High byte is always zero.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The free space on the disk in the specified drive is computed and deposited in the first three bytes of the DMA buffer. The least significant byte is then to be found in the first byte of the buffer, and the next most significant in the second byte. EOS always returns zero in the third byte of the buffer. The number thus specified denotes the amount of unallocated blocks of 128 bytes each. To obtain the free space as measured in Kbytes, divide 8 into the number.

If the drive specified is not yet defined, it is accessed physically to determine its characteristics. Physical errors occurring during this read function are treated according to the current error handling mode.

****
*
* 47 - Chain Program
*
****

On entry: C = 2FH
DE = Purge Flag

On return: (does not return)

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This function is essentially a specific command interpreter start. In the DMA buffer, EOS expects a normal command line, terminated by a zero byte. The command interpreter treats this command line right the same way as if typed in at the keyboard, default drive and user area are set to the values last used by the command interpreter. If the purge flag (register E) is not zero, current drive and user area are retained on program start.

Any errors within the command line are handled as if the command line were type in at the keyboard.

By means of function 1Ø8, values may be passed to the chained-to program.

48 - Flush Data Buffers

On entry: C = 30H
DE = Purge Flag

On return: A = Error Code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
4 = Drive not selectable

Differences to CP/M plus:

The purge flag is not interpreted.

Differences to CP/M 2.2 resp. DiCOS:

New function.

Due to EOS memory management, data buffers are not written to the disks immediately following a disk write command. This function writes all yet unwritten buffers. Physical errors are treated according to the current error handling mode.

```
**************************************************************************
*                                                                        *
*                    49 - Get / Set System Variables                     *
*                                                                        *
**************************************************************************
```

On entry: C = 31H
DE = Pointer to Parameters

On return: HL = Value of Requested System Variable

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

Function 49 allows for accessing the internal EOS system variables' area. DE points to an area with following parameters:

```
DB   OFFSET  ; Address of variable relative to begin of area
DB   CODE    ; Read / write flag
DW   VALUE   ; Byte or word for new value
```

The read/write flag may assume following values:

ØFEH = Set byte
ØFFH = Read byte
ØØØH = Read value

All other values are reserved.

For the permitted values of OFFSET, refer to chapter "System Variables".

WARNING: System variables may be read anytime, but careless writing to a system variable will inevitably cause a system crash!

```
*************************************************************************
*                                                                       *
*                  5Ø - Direct Call of Hardware Interface               *
*                                                                       *
*************************************************************************
```

On entry: C = 32H
DE = Parameters

On return: Registers set to return values

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Function 27 is supported.

Differences to CP/M 2.2 resp. DiCOS:

New function.

As EOS no longer has a CP/M-like BIOS, this function has been introduced to enable application programs emulate the former CP/M - BIOS functions. EOS maintains a jump vector table compatible to the BIOS entry table of DiCOS resp. CP/M 2.2, which is simulated using this function, too.

DE must be loaded with the address of following parameter table:

```
DB    FUNC     ;  'BIOS' function code
DB    AREG     ;  Value for register A
DW    BCREG    ;  Value for register pair BC
DW    DEREG    ;  Value for register pair DE
DW    HLREG    ;  Value for register pair HL
```

For more details on the functions implemented and the values they return, refer to Section B.

59 - Load Overlay

On entry: C = 3BH
DE = Address of FCB

On return: A = Error code
H = Physical error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable
9 = Quotation marks in file name

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

This function enables user programs to invoke the EOS program loader. Normally, the loader is used by the command interpreter only and does not reside in memory. To keep it in memory, function modules must have been loaded, or the program must be provided with an empty program header.

EOS expects an active FCB in DE, i.e. the file must have been opened. Furthermore, the desired loading address has to be passed in bytes 34 and 35 (R0 and R1) in the FCB, which must not be less than 100H. There must be enough space in memory to store the entire file. If both these conditions are not met, EOS returns with register A = 0FFH and H = 0. All other logical error possible correspond to those mentioned at function 20 (Read Sequential).

If the file loaded is of type "PRL", it will be relocated after loading. Function modules found within the file will be activated.

```
*************************************************************************
*                                                                       *
*                    60 - Invoke Function Module                        *
*                                                                       *
*************************************************************************
```

On entry: C = 3CH
DE = Parameters

On return: Free error code

Possible logical errors:

Function dependent.

Possible physical errors:

Function dependent.

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

Function 60 allows to invoke active function modules for initialization, parameter passage and other purposes. DE points to a table containing the following parametes:

```
DB    SFUNC  ; Special function
DW    NPARAM ; Number of parameters
DW    PARAM1 ; Parameter 1
   :
   :
DW    PARAMn ; Parameter n
```

Special functions 128 to 255 are reserved for EOS.

If none of the functions modules felt being addressed, EOS returns 0FFH in A and 00H in H.

****
*
* 64 - Set Error Handling Mode *
*
****

On entry: C = 40H
E = 0 or 0FFH

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function.

This function has the same effect as function 45, except that all other values than 0FFH will cause a return to direct error handling.

Function 64 has been retained for compatibility to DiCOS.

****
*
* 65 - Change Process Number *
*
****

On entry: C = 41H
E = Process Number

On return: A = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function.

This function has been retained to preserve DiCOS compatibility, however, it has no effect any more.

****
*
* 66 - Dump Memory
*
****

On entry: C = 42H
DE = Parameters

On return: A = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function.

This function has been retained to preserve DiCOS compatibility, however, it has no effect any more.

67 - Test for Break Character

On entry: C = 43H
DE = Character or ØFFH

On return: A = Ø or ØFFH

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function.

The character passed in register E is checked whether it is one of either BREAK characters (Control-C or the user definable). If so, A is set to ØFFH, otherwise to Ø.

If E contains ØFFH on entry, the keyboard will be checked previously. When there is a character pending, this will be checked.

If the check for BREAK characters has been suppressed by function 109, A is always returned set to Ø.

****
*
* 68 - Execute File
*
****

On entry: C = 44H
DE = Address of FCB

On return: A = error code

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function, but DE = Ø is no longer permitted.

The file denoted by the FCB is executed, i.e. keyboard input is redirected to the file. The file may contain several control characters. For more detailed information, refer to the $DO.COM program.

If the file could not be executed, either because it has not been found or a physical error occurred, a value of ØFFH is returned in A, otherwise Ø. When an error is detected during executing the file, input will be directed back to the keyboard again, without any comment. The same will happen when the end of file is encountered.

If the file attribute bit f4' is set in the FCB, the file will be erased after execution.

Under EOS, this function has been inplemented by a function module named $DO.RSX. If a program needs this service, it can be linked to that program by the $MOD utility.

69 - Background Printing of a File

On entry: C = 45H
DE = Address of FCB

On return: A = error code

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function, but DE = Ø is no longer permitted.

The file denoted by the FCB is printed in background mode.

If the file could not be opened, either because it has not been found or a physical error occurred, a value of ØFFH is returned in A, otherwise Ø. When an error is detected during printing, the print process is aborted. The same will happen when the end of file is encountered.

If the file attribute bit f4' is set in the FCB, the file will be erased after printout is completed.

Under EOS, this function has been inplemented by a function module named $SPOOL.RSX. If a program needs this service, it can be linked to that program by the $MOD utility.

```
**************************************************************************
*                                                                        *
*                    70 - Control Background Print                       *
*                                                                        *
**************************************************************************
```

On entry: C = 46H
E = Control Code

On return: A = Error Code

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

Old DiCOS function.

A background printing process started by function 69 can be controlled using this function. Following control codes are defined:

- 0 = Check if background printing is active.
- 1 = Restart background printing. The file is printed again from its beginning.
- 2 = Abort background printing. The file will be erased if attribute bit f3' is set.
- 3 = Resume background printing. A background printing process interrupted by code 4 is continued.
- 4 = Interrupt background printing. Printing is stopped until it is resumed or aborted by one of the codes 1, 2, or 3.

Under EOS, this function has been implemented by a function module named $SPOOL.RSX. If a program needs this service, it can be linked to that program by the $MOD utility.

71 - Check Drive Ready

On entry: C = 47H
DE = Drive Number

On return: A = Ø or ØFFH
H = Bit Vector

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The drive denoted by register E is checked if it is ready for data transfer. A value of Ø corresponds to drive A, 1 to drive B, and so forth up to 15 for drive P. If the drive is ready, A is set to ØFFH and the bits in H are set as follows:

Bit 7 set = Data carrier cannot be removed.
Bit 6 set = Drive contains two-sided diskette.

If the drive is not ready, A is set to Ø. If the drive parameters are undefined, A is set to 7.

It should be noted that no physical drive access is done, so the disk format will not be determined. Hence, subsequent read/write operations can lead to errors.

****
*
* 72 - Get / Set Version Number *
*
****

On entry: C = 48H
DE = Version Number or ØFFFFH

On return: A = Version Number, if requested.

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The system version number returned by function 12 is set to the value passed in register E. It becomes thus possible to run programs which expect a specific system version.

If DE is set to ØFFFFH, the current version number is returned in register A.

73 - Get / Set Printer timeout

On entry: C = 49H
DE = timeout interval or ØFFFFH

On return: A = timeout interval, if requested.

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The timeout interval which has to elaspe before the system requests operator intervention for non-ready output devices is set to the data passed in E. Register E can be set to any value from Ø to 255, where Ø is interpreted as 256.

If DE is set to ØFFFFH, the current timeout interval is returned in register A.

For details about the timeout interval, refer to "Device assignment".

74 - Get / Set Tab Stops

On entry : C = 4AH
DE = Tab stop or ØFFFFH

On return: A = Tab stop, if requested.

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The tab stop used with functions 2, 9, and 111 is set to a new value passed in E. Register E may contain only one of the values 2, 4, 8, 16, or 32. Any other value will cause a confusing screen image.

If DE is set to ØFFFFH, the current tab stop is returned in register A.

On system start, the tab stop is set to 8.

****
*
* 75 - Get / Set BREAK Characters
*
****

On entry : C = 4BH
DE = BREAK character or ØFFFFH

On return: A = BREAK character, if requested.

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The user definable second BREAK character is set to the character passed in register E. If E is zero, the second BREAK character is disactivated.

If DE is set to ØFFFFH, the current second BREAK character is returned in register A.

On system start, no second BREAK character is defined.

76 - Get / Set EOS Flags

On entry : C = 4CH
DE = Flags or ØFFFFH

On return: A = Flags, if requested.

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

EOS maintains a flag byte controlling various functions, which can be read and set by means of this function. The bits of this flag byte have the following meanings:

Bit 7 - reserved for CP/M plus compatibility -
Bit 6 - reserved for CP/M plus compatibility -

Bit 5 ON = Function 1Ø uses the screen editor.
OFF = Function 1Ø uses the EOS mini editor.
Preset state: OFF

Bit 4 ON = When a BREAK character is typed, EOS asks the operator if the program shall be aborted.
OFF = Program abortion on BREAK character typed in.
Preset state: OFF

Bit 3 ON = On each system call, the keyboard is checked for a BREAK character typed in, and if so, the program is aborted.
OFF = Normal behaviour of EOS.
Preset state: OFF

Bits Ø, 1, and 2 are reserved. See also the chapter on the flag byte.

If DE is set to ØFFFFH, the current flag byte is returned in register A.

****
*
* 98 - Release Data Blocks
*
****

On entry: C = 62H
DE = ---

On return: A = Ø
HL = Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Not implemented.

Differences to CP/M 2.2 resp. DiCOS:

New function.

This function has not been implemented, because EOS has another memory management algorithm than CP/M. A call of this function will return the above mentioned values.

```
**************************************************************************
*                                                                        *
*                    99 - Truncate File                                  *
*                                                                        *
**************************************************************************
```

On entry: C = 63H
DE = Address of FCB

On return: A = Error Code
H = Physical Error Code

Possible logical errors:

- see below -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
3 = File is write protected
4 = Drive not selectable
9 = Question marks in filename

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

DE points to the FCB of a file which must not be opened. Bytes 34 to 36 of the FCB denote a record number which will be declared as the last block of the file concerned. All data in records past the specified record number are lost. The associated disk space is released and becomes free for other purposes.

The record number is stored in reverse order, i.e. byte 34 is the least significant byte of the number, byte 35 the next one, and byte 36 the most significant (always Ø under EOS).

If the file was successfully truncated, A and H are set to Ø upon return. In case of logical errors, A is set to ØFFH and H is zero. Possible reasons for logical errors are:

- File not found
- Specified record number past end of file
- Specified record number in a region of the file which has not yet been written to by random mode disk writes.

Physical errors are treated according to the current error handling mode.

```
*************************************************************************
*                                                                       *
*                  100 - Write Directory Label                          *
*                                                                       *
*************************************************************************
```

On entry: C = 64H
DE = Address of FCB

On return: A = Error Code
H = Physical Error code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
2 = Drive is write protected
4 = Drive not selectable

Differences to CP/M plus:

Passwords are ignored.

Differences to CP/M 2.2 resp. DiCOS:

New function.

A new directory label is written to the drive specified by the FCB. The filename in the FCB becomes the future data carrier (disk) name. Byte 13 of the FCB (extent byte) holds a bit vector with following information:

Bit 6 = Write time-stamps on each file access.
Bit 5 = Write time-stamps on file alterations.
Bit 4 = Write time-stamps on creation of new files.

All other bits are reserved.

Bit 4 and bit 6 cannot be set simultaneously, as there is only one data field for both types of time-stamps. Attempts to create such a label will be refused.

If writing time-stamps is requested, the directory has to be prepared to accept time-stamps by the utility program $INITDIR. Otherwise, no such directory label will be created.

After a successful creation resp. alteration of the directory label, A and H are returned with zero values. If due to missing time-stamp entries or incompatible kinds of time stamps, no new label has been written, A is set to ØFFH and H is Ø. Physical errors are treated according to the current error handling mode.

101 - Get Directory Label Data Byte

On entry: C = 65H
E = Drive Number

On return: A = Data Byte or Error Code
H = Physical Error Code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The data byte of the directory label of the drive specified by register E is returned in A. On entry, a value of 0 in E denotes drive A, 1 drive B and so on. The data byte itself is byte 13 in the directory label entry, containing the following bits:

Bit 6 = Write time-stamps on each file access.
Bit 5 = Write time-stamps on file alterations.
Bit 4 = Write time-stamps on creation of new files.
Bit 0 = Directory label exists.

If no directory label exists, A is returned with a zero value, otherwise it holds the data byte. In both cases, H is set to 0. Physical errors are treated according to the current error handling mode.

****
*
* 102 - Read Time Stamps
*
****

On entry: C = 66H
DE = Address of FCB

On return: A = Error Code
H = Physical Error Code

Possible logical errors:

- none -

Possible physical errors:

1 = Read/write error
4 = Drive not selectable

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The file denoted by the FCB is searched for on the drive specified. Its associated time stamps are returned in the FCB as follows:

Byte 13	=	Password byte, always 0 under EOS.
Bytes 25 - 28	=	Time stamp of creation or last access.
Bytes 29 - 32	=	Time stamp of last file alteration.

If there are no time stamps, the fields mentioned above are zero filled. For the format of time stamps, refer to the corresponding chapter.

On a successful completion of this function, A and H are both returned with a zero value. If the file has not been found, A is set to 0FFH, whereas H is still zero. Physical errors are treated according to the current error handling mode.

****
*
* 103 - Write XFCB
*
****

On entry: C = 67H
DE = Address of XFCB

On return: A = 0FFH
H = 0

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Function not implemented.

Differences to CP/M 2.2 resp. DiCOS:

New function.

Under CP/M plus, this function is used to create passwords for a file. Because EOS does not support passwords, the call is ignored and the registers are set as stated above.

****
*
* 1Ø4 - Set Date
*
****

On entry: C = 68H
DE = Address of Date Fields.

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

By means of this function, the system date can be set to any desired value. DE points to a memory area containing following data:

```
DW   DATE   ; Julian Date. Day number 1 is January 1st, 1978.
DB   HOUR   ; Hour, BCD format.
DW   MINUTE ; Minute, BCD format.
```

The seconds are always set to Ø upon a call of function 1Ø4.

```
**************************************************************************
*                                                                        *
*                    1Ø5 - Get Date                                      *
*                                                                        *
**************************************************************************
```

On entry: C = 69H
DE = Address of Date Fields.

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The internal system clock/calendar is read and the data deposited at the memory area pointed to by DE in the following manner:

```
DW   DATE   ; Julian Date. Day number 1 is January 1st, 1978.
DB   HOUR   ; Hour, BCD format.
DW   MINUTE ; Minute, BCD format.
```

The seconds are returned in register A as two BCD digits.

106 - Set Password

On entry: C = 6AH
DE = Address of Password.

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

Function not implemented.

Differences to CP/M 2.2 resp. DiCOS:

New function.

Under CP/M plus, this function sets a standard password. Because EOS does not support passwords, the call is simply ignored.

```
**************************************************************************
*                                                                        *
*                    107 - Get Serial Number                             *
*                                                                        *
**************************************************************************
```

On entry: C = 6BH
DE = Address of Field to store Serial Number

On return: (Serial Number stored)

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The serial number of the system kernel is stored in the memory area pointed to by DE in the following manner:

```
     DB   OEM        ; Serial number of manufacturer.
     DW   USER       ; Serial number of end user.
```

****
*
* 108 - Get / Set Program Return Code
*
****

On entry: C = 6CH
DE = Return Code or 0FFFFH

On return: HL = Return Code

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

Function 108 provides for a small, limited commucation between programs. However, the programs must be loaded by function 47, as the command interpreter always resets the program return code to 0. The system kernel also sets certain return codes. Following values are permitted:

```
0000 - FEFF = Successful program execution.
FF00 - FF7F = Not successful program execution.
FF80 - FFFC = - reserved -
FFFD        = Program aborted due to physical error.
FFFE        = Program aborted due to BREAK command by operator.
```

If DE is set to 0FFFFH on entry, EOS returns the current return code.

****
*
* 109 - Get / Set Console Mode *
*
****

On entry: C = 6DH
DE = 0FFFFH (for reading) or Bit Vector

On return: HL = Console Mode

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The reaction of EOS can be influenced in various ways by this function along with function 76, which provides further ways of system control. The bit vector passed in DE has following meanings, where bit 0 is the least significant bit of register L:

Bit 0 = ON : Function 11 returns another value than zero only when a BERAK character has been typed.
OFF: Function 11 returns a non-zero value on any character typed in.

Bit 1 = ON: It is not possible to interrupt screen output by typing control-S. Control-S is recognized as input.
OFF: Screen output can be interrupted by typing control-S and resumed by typing control-Q or another control-S.

Bit 2 = ON: Tab stops are not expanded. It is not possible to switch the printer on/off by typing control-P.
OFF: Tab stops are expanded according to the current setting. Printer control by typing control-P is active.

Bit 3 = ON: BREAK characters are invalidated. Programs cannot be aborter by typing a BREAK character.
OFF: BREAK characters are valid.

Bits 8 and 9 are used for keyboard redirection, they control keyboard state:

ØØ = Keyboard state requests are ingnored, except when two state requests follow immediately upon one another. In this case, the second state request returns 1.

Ø1 = Keyboard state requests always return ØØH.

1Ø = Keyboard state requests always return Ø1H.

11 = Redirection is ignored.

11Ø - Get / Set String Delimiter

On entry: C = 6EH
DE = ØFFFFH (for reading) or E = String Delimiter

On return: A = String Delimiter

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

The string delimiter used with function 9, normally the '$' character, can be arbitrarily defined. If ØFFFFH is passed in DE, the current string delimiter is returned in register A.

```
**************************************************************************
*                                                                        *
*                  111 - Print Block to Console                          *
*                                                                        *
**************************************************************************
```

On entry: C = 6FH
DE = Pointer to parameter block

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

DE points to an area containing the following parameters:

```
     DW   START    ;  Starting address of block to be printed.
     DW   SIZE     ;  Length of block in bytes.
```

The full block of memory is printed on the screen. As EOS uses function 2 internally, all control features described there are active, dependent on the current console mode.

112 - Print Block to Printer

On entry: C = 70H
DE = Pointer to parameter block

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

- none -

Differences to CP/M 2.2 resp. DiCOS:

New function.

DE points to an area containing the following parameters:

```
DW   START   ;  Starting address of block to be printed.
DW   SIZE    ;  Length of block in bytes.
```

The full block of memory is printed on the printer.

```
**************************************************************************
*                                                                        *
*                  115 - Access built-in Graphics                        *
*                                                                        *
**************************************************************************
```

On entry: C = 73H
DE = Pointer to Parameter Area

On return: ---

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

New function.

Differences to CP/M 2.2 resp. DiCOS:

New function.

Function 115 provides for direct control of the built-in graphic functions. Following data have to be set up in the area where DE points to:

```
     DB   OPCODE ;  Graphic operation code
     DW   PARAM1 ;  Parameter 1
        :
        :
     DW   PARAMn ;  Parameter n
```

The number of parameters required depends on the specific graphic function called. For details, refer to section C - "Graphics interface".

152 - Parse String and Construct FCB

On entry: C = 98H
DE = Pointer to Parameter area

On return: HL = pointer or Ø

Possible logical errors:

- none -

Possible physical errors:

- none -

Differences to CP/M plus:

No errors can occur.

Differences to CP/M 2.2 resp. DiCOS:

New function.

The parameter area where DE points to must contain following two addresses:

```
DW   STRING ;  Address of string to be parsed
DW   FCB    ;  Address of FCB to be made up
```

The string specified is parsed for a valid filename and an FCB is constructed accordingly. The string may contain up to 128 characters. The FCB must be 36 bytes long. The parsing process obeys to the following rules:

Spaces and tabs are skipped. If a colon is found, the immediately preceding character will be used as drive indicator. Filename and type are taken over as specified, the filename must be separated from the file type by a period. If a semicolon follows the filename, the subsequent string is used as password. If filename or password are longer than 8 or the file type longer than 3 characters respectively, the remainder of the partial string is ignored. All letters are converted to upper case.

The FCB ist set up as:

```
Byte   Ø      = Drive indicator or Ø
Bytes  1 -  8 = Filename, padded with spaces
Bytes  9 - 11 = File type, padded with spaces
Bytes 12 - 16 = Filled with zeros
Bytes 17 - 24 = Password, padded with spaces
Bytes 25 - 32 = Filled with zeros
```

If name, type or password are not specified, the corresponding FCB data fields are filled with spaces.

The following list shows which characters are recognized as delimiters:

```
ASCII control characters
Space
ASCII null (ØØH)
; except between type and password
=
<
>
. (period) except between name and type
: except between drive and name
, (comma)
\
[
]
```

After the FCB is completed, EOS scans the subequent part of the string, skipping spaces and tabs. On return, HL is set according to the following condition:

Char. encountered	Value in HL
ØØH or CR	Ø to denote end
delimiter	address of delimiter
any other	address of first character past the last one used for the FCB information.

Section B

Hardware Interface

B.1. Preview

As any other operating system, EOS needs a sufficiently well defined software interface for operating the MZ-3500 hardware. This kind of interface provides for a number of functions, comparable to the EOS functions themselves. By means of these functions, characters are sent and received, disks are read and written, and various other tasks are performed. Many of these functions are of interest to users who want to write hardware related programs and hence documented here. Screen and graphics control are described in sections of their own in order to prevent this section from becoming too large and unhandy.

This section contains detailed information on:

- Disk parameter block (DBP) structure
- Calling hardware functions
- a table of hardware driver error messages

B.2. Calling Hardware Functions

All calls of hardware functions are done using EOS function 50. When calling this system function, the following data are to be passed in the CPU registers:

Register C: 32H
Register DE: Pointer to a table of this kind:

```
        DB      FUNC            ; Hardware function code
        DB      AREG            ; Data for register A
        DW      BCREG           ; Data for register pair BC
        DW      DEREG           ; Data for register pair DE
        DW      HLREG           ; Data for register pair HL
```

Prior to calling the specified hardware driver function, EOS loads the CPU registers with the data from the parameter table. All results are returned in register pair HL. As a convenience, the contents of register L are copied into register A, too. Function 25 (Move Memory Block) represents an exception to this rule, as it returns all register pairs BC, DE, and HL with changed contents.

Let us assume You want to display a character on the screen directly, bypassing EOS. Then, You would have to write a piece of code similar to this:

```
        LD      C,50            ; EOS function code
        LD      DE,REGSET       ; Register settings
        CALL    5               ; Execution
          :
          :
REGSET: DB      4               ; Hardware function code
        DB      0               ; A register
        DW      CHAR            ; Character to be displayed in BC
        DW      0               ; DE register
        DW      0               ; HL register
```

B.3. The Hardware Entry Jump Table

The operating systems CP/M and DiCOS are also provided with a facility to access the hardware interface, implemented as a jump vector table. A jump to the warm star entry is deposited at address Ø, thus enabling the user program to obtain the address of the vector table. To maintain compatibility to CP/M 2.2 and DiCOS, a table of this kind is present in EOS, too. On the MZ-35ØØ, it begins at address ØF5ØØH and contains the vectors as specified below:

```
        JP      FUNCØ           ; Cold start entry, DO NOT USE!
WARM:   JP      FUNC1           ; Warm start entry
        JP      FUNC2           ; Keyboard state check
        JP      FUNC3           ; Keyboard input
        JP      FUNC4           ; Screen output
        JP      FUNC5           ; Printer output
        JP      FUNC6           ; Aux. device output
        JP      FUNC7           ; Aux. device input
        JP      FUNC8           ; Move head to track Ø
        JP      FUNC9           ; Select drive
        JP      FUNC1Ø          ; Set track
        JP      FUNC11          ; Set sector
        JP      FUNC12          ; Set DMA address
        JP      FUNC13          ; Read disk
        JP      FUNC14          ; Write disk
        JP      FUNC15          ; Check printer state
        JP      FUNC16          ; Sector translation
```

It shall be mentioned that a jump to the warm start entry causes the command interpreter and the program loader to be reloaded.

Address Ø contains a jump instruction:

```
        JP      WARM            ; corresponding ØF5Ø3H on the MZ-35ØØ
```

For example, let us assume we want to display a character on the screen:

```
        LD      C,CHAR          ; Character to be displayed
        LD      HL,(1)          ; Destination of jump instruction
        LD      L,12            ; Offset of screen output entry
        JP      (HL)            ;  and jump there.
```

B.4. Hardware Drivers' Error Messages

There are two kinds of error messages, the first one relating to read and write operations on magnetic data carriers. It is formatted as follows:

```
(ttssm) Message
```

where the characters denote:

```
tt - Track accessed
ss - Sector accessed (both these values are in hex)
m  - Mode, R = Read, W = Write
```

The messages possible are:

1) Sector address not found
2) Sector not found
3) Drive not ready
4) Checksum error
5) End of track encountered
6) Erroneous parameters
7) System error
8) Internal error

The error messages mean:

1) No address marks have been found on the data carrier, e. g. due to another recording format.

2) The desired sector was not found on the track specified.

3) The drive is not ready to accept or transmit any data.

4) The sector read has an erroneous checksum, it contains bad data.

5) An end-of-sector condition was encountered on a multisector disk read. This error occurs only when the disk parameter block is improperly set up.

6) Erroneous parameters, e.g. track number specified was too large.

7) System error. All other disk read/write errors are collected here.

8) Generally, a "lethal" error. Because of some strange reason, EOS ran out of control. The only remedy is a system cold start. This kind of error should never occur.

The second kind of error messages refers to non-ready output devices and looks like this:

Device XXXXXX not ready

where XXXXXX denotes the name of the device as defined in the device table. The message will appear after timeout elapsed (the timeout interval can be set by EOS function 73). If at this moment, a character has been typed at the keyboard, output is aborted and a system restart occurs. When there is no keyboard input pending, the time-out process repeats. For the screen, no time-outs are performed.

B.5. Firmly Installed Drives

This term refers to all drives which are installed fixed in the computer system, mainly hard disks and the RAM disk. These drives are not affected by EOS function 13 (reset disk system). Generally , resetting drives is necessary only for the purpose of re-defining the drive characteristics after a media change and hence, unnecessary with firmly installed drives. Such a unit is defined at the first physical access and remains so. Should it become necessary to reset even a fixed installed drive, this can be achieved by EOS function 37 (reset drive).

The system recognizes a firmly installed drive by the SKEW byte in its DPB (see below). With such drives, the most significant bit in the skew byte is set.

B.6. The Skew Factor

The skew factor determines how logical sectors are mapped to physical sectors on disks. To gain a high disk I/O troughput, EOS does not access physically successive sectors. After a specific sector has been read, the next sector (may be several) is skipped so that EOS has some time to process the data read. The distance between two sectors which are read in sequence in called "skew factor". For example, let the skew factor be 2, then the sectors 1, 3, 5, and so forth are transferred from disk when reading logically successive sectors.

B.7. The Disk Parameter Block

For each drive connected to the system, a so-called Disk Parameter Block (DPB) is defined. A DPB contains all data related to disk recording format. For example, a DBP of its own is necessary for each recording format of an 8-inch floppy disk. The starting address of a specific DPB can be obtained by EOS function 31. Furthermore, the start address of the DPB of the current drive can directly be read from the system variables' area. However, it is recommended to avoid doing so, as EOS does not guarantee that the information stored there is in accordance with the DPB of the current drive.

Disk parameter blocks are structured as described below:

Bytes 1 - 2 : Setors Per Track (SPT). This number specifies how many records of 128 bytes each fit into one track. "Track" does not necessarily refer to physical tracks. E.g., on double sided disks, the corresponding tracks of the top and rear side are concatenated to form one logical track.

Byte 3 : Block Shift Factor (BSF). The BSF specifies how many bits a logical block number (which addresses 128 bytes) has to be shifted right to obtain the number of the allocation cluster. Cluster size and BSF are related as follows:

Cluster size	Shift Factor
1 Kbytes	3
2 Kbytes	4
4 Kbytes	5
8 Kbytes	6
16 Kbytes	7

Byte 4 : Block Mask (BM). This byte holds a mask which is laid over the block number to obtain the block offset within the allocation cluster. The bits retained after masking are just the ones lost when shifting the block number according to the BSF. The BSM depends of the cluster size as described below:

Cluster size	Shift Factor
1 Kbytes	7 = Ø7H
2 Kbytes	15 = ØFH
4 Kbytes	31 = 1FH
8 Kbytes	63 = 3FH
16 Kbytes	127 = 7FH

Byte 5 : Extent Mask (EM). When searching in the directory, the extent mask is laid over the extent byte to obtain the correct directory entry for the desired extent. The EM must be adapted to the number of clusters defined. If there are more than 256 clusters, one cluster occupies two bytes in the allocation list of the entry. Hence, less clusters will fit in the allocation list, making the EM smaller. The table below explains the relations.

Cluster size	small disks	large disks
1 Kbytes	Ø	undefined
2 Kbytes	1	Ø
4 Kbytes	3	1
8 Kbytes	7	3
16 Kbytes	15	7

Bytes 6 - 7 : Highest cluster number. This number is one less than the number of clusters available on disk (the cluster numbers begin at Ø).

Bytes 8 - 9 : Number of last directory entry. This number is one less thas the number of directory entries available on disk. A directory entry occupies 32 bytes, thus 4 entries fit into one 128-byte block. Consequently, a cluster of 1 Kbyte holds up to 32 entries, a cluster of 2 Kbytes 64 entries, and so forth.

Bytes 10 - 11 : Pre - Allocation Map. A bit vector is stored in these two bytes which is used for initializing the allocation vector. By means of the pre-allocation map, the clusters required for the directory are reserved.

Bytes 12 - 13 : Checksum Vector Size. The checksum vector is a table of checksums (one byte per 128-bytes block) referred to by EOS on each directory operation. If one or more checksums change from one directory access to another, this is regarded as a data carrier change detected and the disk is automatically write protected. This mechanism is meaningful for removable data media only. Non-interchangeable data carriers have no checksum vectors, the field has a zero value.

Bytes 14 - 15 : Number of reserved tracks. These bytes denote the first track used for storing data, thus providing for a means to reserve the outermost tracks for the operating system.

Byte 16 : Sector Size, denoting the physical sector size of the data carrier in units of 128 bytes. It must not necessarily be in accordance with the actual physical sector size, but specifies how many bits a logical sector number has to be shifted right to obtain the physical sector number:

Sector Size	Shift Factor
128 bytes	0
256 bytes	1
512 bytes	2
1024 bytes	3
2048 bytes	4
4096 bytes	5

Byte 17 : Sector Mask. This mask is laid over the logical sector number (addresing 128 bytes) to obtain its offset within a physical sector. It retains just the bits lost when shifting the logical sector number:

Sector size	Shift Mask
128 bytes	0
256 bytes	1
512 bytes	3
1024 bytes	7
2048 bytes	15
4096 bytes	31

Byte 18 : Skew Factor. If this byte is set to 1, no skew factor is applied to the disk concerned. A zero value means that the sectors are stored on disk in a skewed manner determined via a table. This table, however, is not accessible by user programs.

B.8. Table of Hardware Driver Calls

No	Function	Parameters	Data returned
Ø	System Initialization	---	---
1	System Re-Initialization	---	---
2	Check Concols State	---	AL = Ø / character
3	Console Input	---	AL = character
4	Console Output	C = character	---
5	Printer Output	C = character	---
6	Auxiliary Device Output	C = character	---
7	Auxiliary Device Input	---	AL = character
8	Move Disk Head to Track Ø	---	---
9	Select Drive	C = drv, E = bit	HL = Ø / not Ø
1Ø	Set Track	BC = track	---
11	Set Sector	BC = sector	---
12	Set DMA Address	BC = address	---
13	Read Disk Block	---	AL = Ø / 1
14	Write Disk Block	C = write code	AL = Ø / 1 / 2
15	Check Printer State	---	AL = Ø / ØFFH
16	Sector Translation	BC = sector	HL = sector
17	Check Console Output State	---	AL = Ø / ØFFH
18	Check Aux. Dev. Input State	---	AL = Ø / character
19	Check Aux. Dev. Output State	---	AL = Ø / ØFFH
2Ø	Return Device Table Address	---	HL = address
21	Initialize Device	C = device number	---
22	Check Drive Ready	C = drive	HL = check result
23	Activate Screen Editor	C = col, DE = addr	---
24	Flush Sector Buffers	---	AL = Ø / 1 / 2
25	Move Memory Block	registers preset	registers changed
26	Get/Set Calendar/Clock	C = flag, DE = addr	---
27	Invoke Graphics	BC = parameters	---
28	Hex. Output	BC = word f. output	---

```
**************************************************************************
*                                                                        *
*                    Ø - System Initialization                           *
*                                                                        *
**************************************************************************
```

On entry: ---

On return: ---

This is the cold start entry of the hardware interface. All necessary initializations are carried out, including memory management and device drivers and the command-interpreter is copied into the RAM-disk.

It it not recommended to make use of this function, as a number of the initialization routines should or even must run only once. For instance, the command interpreter cannot be copied twice into the RAM-disk, because one copy already exists.

```
**************************************************************************
*                                                                        *
*                    1 - System  Re-Initialization                       *
*                                                                        *
**************************************************************************
```

On entry: ---

On exit : ---

On each system warm start (i.e. by calling EOS function Ø or by a jump to address Ø) this section of the hardware interface is invoked. Yet unwritten data buffers are flushed, and the device drivers are brought into a well-defined state.

This function may be called by a user program, although it is of highly doubtful effect.

Example:

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
            :
            :
PARAMS:   DB        1             ; Hardware function
          DB        Ø
          DW        Ø
          DW        Ø
          DW        Ø
```

```
*************************************************************************
*                                                                       *
*                    2 - Check Console State                            *
*                                                                       *
*************************************************************************
```

On entry: ---

On exit : AL = State Code

All devices mapped to the console input via the CONIN: bit vector are checked for a character pending. If at least one of these devices has a character ready for input, registers A and L are assigned the character which would be read, otherwise a value of ØØH.

Example:

```
            LD          C,5Ø          ; EOS function code
            LD          DE,PARAMS     ; Parameters
            CALL        5             ; Call EOS
            OR          A             ; Ready for input
            JP          NZ,INPRDY
              :
              :
PARAMS:     DB          2             ; Hardware function No.
            DB          Ø
            DW          Ø
            DW          Ø
            DW          Ø
```

```
*************************************************************************
*                                                                       *
*                    3 - Read Console Character                         *
*                                                                       *
*************************************************************************
```

On entry: ---

On exit : AL = Character

All devices mapped via the CONIN: bit vector to the console input channel are checked for a pending input. The first device found is then read, and the character is stored in registers A and L. No transformation will occur, i.e. the 8-bit character is returned as read from the device.

The driver waits until there is at least one device ready for input.

Example:

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
          LD        (CHAR),A      ; Store character
            :
            :
PARAMS:   DB        3             ; Hardware function No.
          DB        Ø
          DW        Ø
          DW        Ø
          DW        Ø
```

```
**************************************************************************
*                                                                        *
*                    4 - Write Console Character                         *
*                                                                        *
**************************************************************************
```

On entry: C = Character

On exit : ---

The character passed in register C is sent to all devices mapped via the CONOUT: bit vector to the console output channel. The character is not masked or converted anyway. The system waits until all devices affected have accepted the character.

If a device is not ready to receive a character, the system waits until the timeout interval (definable by EOS function 73) has elapsed. Then, an error message is displayed on the screen denoting the faulty device. If at this moment hardware function 2 returns a non-zero value, the character pending is read and output will be aborted, i.e. a system restart is performed. Otherwise, the wait cycle repeats.

Output device 0, however, is not subjected to timeouts, because it is defined as monitor screen under EOS.

Example: Display the character "*" on the screen

```
            LD      C,50        ; EOS function code
            LD      DE,PARAMS   ; Parameters
            CALL    5           ; Call EOS
              :
              :
PARAMS:     DB      4           ; Hardware function No.
            DB      0
            DW      '*'         ; character in register C
            DW      0
            DW      0
```

```
**************************************************************************
*                                                                        *
*                    5 - Print Character                                 *
*                                                                        *
**************************************************************************
```

On entry: C = Character

On exit : ---

The character passed in register C is sent to all devices mapped via the LSTOUT: bit vector to the printer. The character is not masked or converted anyway. The system waits until all devices affected have accepted the character.

If a device is not ready to receive a character, the system waits until the timeout interval (definable by EOS function 73) has elapsed. Then, an error message is displayed on the screen denoting the faulty device. If at this moment hardware function 2 returns a non-zero value, the character pending is read and output will be aborted, i.e. a system restart is performed. Otherwise, the wait cycle repeats.

Output device Ø, however, is not subjected to timeouts, because it is defined as monitor screen under EOS.

Example: Send a form feed to the printer

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
            :
            :
PARAMS:   DB        5             ; Hardware function No.
          DB        Ø
          DW        ØCH           ; character in register C
          DW        Ø
          DW        Ø
```

```
*************************************************************************
*                                                                       *
*                  6 - Write Character to Auxiliary Device              *
*                                                                       *
*************************************************************************
```

On entry: C = Character

On exit : ---

The character passed in register C is sent to all devices mapped via the AUXOUT: bit vector to the auxiliary device. The character is not masked or converted anyway. The system waits until all devices affected have accepted the character.

If a device is not ready to receive a character, the system waits until the timeout interval (definable by EOS function 73) has elapsed. Then, an error message is displayed on the screen denoting the faulty device. If at this moment hardware function 2 returns a non-zero value, the character pending is read and output will be aborted, i.e. a system restart is performed. Otherwise, the wait cycle repeats.

Output device Ø, however, is not subjected to timeouts, because it is defined as monitor screen under EOS.

Example: Send a question mark to the auxiliary device

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
            :
            :
PARAMS:   DB        6             ; Hardware function No.
          DB        Ø
          DW        '?'           ; character in register C
          DW        Ø
          DW        Ø
```

```
*************************************************************************
*                                                                       *
*                  7 - Read Character from Auxiliary Device             *
*                                                                       *
*************************************************************************
```

On entry: ---

On exit : AL = Character

All devices mapped via the AUXIN: bit vector to the auxiliary input channel are checked for a pending input. The first device found is then read, and the character is stored in registers A and L. No transformation will occur, i.e. the 8-bit character is returned as read from the device.

The driver waits until there is at least one device ready for input.

Example:

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
            :
            :
PARAMS:   DB        7             ; Hardware function No.
          DB        Ø
          DW        Ø
          DW        Ø
          DW        Ø
```

```
************************************************************************
*                                                                      *
*                  8 - Move Drive Head to Track ØØ                     *
*                                                                      *
************************************************************************
```

On entry: ---

On exit : ---

The read/write head of the currently selected drive is returned to track zero. This function call, however, does not perform a physical disk access, only internal variables are set.

Example:

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
            :
            :
PARAMS:   DB        8             ; Hardware function No.
          DB        Ø
          DW        Ø
          DW        Ø
          DW        Ø
```

```
*************************************************************************
*                                                                       *
*                      9 - Select Drive                                 *
*                                                                       *
*************************************************************************
```

On entry: C = Drive Number
E = Access Bit

On exit : HL = Error code

The drive passed in register C is defined as reference drive for all future disk read/write operations. A value of "Ø" in C refers to drive A, a "1" to drive B and so forth up to "15" for drive P. The least significant bit in register E specifies whether this is the first access to that drive; if it is reset, the drive parameters have to be re-determined. In this case, an eventual disk access is necessary to determine the recording format. If the "first-access" bit is set, only some internal variables are assigned new values, no disk access will occur.

After a successful disk select operation, HL is returned with a non-zero value. If the desired drive is not defined or an error was encountered during determination of the drive parameters, HL is zero on return.

The access bit in register E is an enhancement over CP/M 2.2 and DiCOS.

Example:

Select drive B: without re-determining the drive parameters

```
            LD      C,5Ø            ; EOS function code
            LD      DE,PARAMS       ; Parameters
            CALL    5               ; Call EOS
            LD      A,H
            OR      L               ; Check if o.k.
            JP      Z,SELERR        ; Zero = no success
              :
              :
PARAMS:     DB      9               ; Hardware function No.
            DB      Ø
            DW      'B'-'A'         ; Drive B = number 1
            DW      1               ; prevent redetermination
            DW      Ø
```

```
**************************************************************************
*                                                                        *
*                    1Ø - Set Track                                      *
*                                                                        *
**************************************************************************
```

On entry: BC = Track Number, counting from Ø upwards

On exit : ---

The number passed in BC is stored as the track number for the next disk access. The track numbers count from Ø upwards.

Example: Set track 3 for next disk access

```
            LD          C,5Ø          ; EOS function code
            LD          DE,PARAMS     ; Parameters
            CALL        5             ; Call EOS
              :
              :
PARAMS:     DB          1Ø            ; Hardware function No.
            DB          Ø
            DW          3             ; Track 3 in BC
            DW          Ø
            DW          Ø
```

```
**************************************************************************
*                                                                        *
*                    11 - Set Sector Number                              *
*                                                                        *
**************************************************************************
```

On entry: BC = Sector Number, beginning from Ø

On exit : ---

The number passed in BC is stored as sector number for the next disk access. The sector numbers count from Ø upwards. In most cases, this is not the physical sector but a "logical" block of 128 bytes within the track defined by hardware function 1Ø. Under certain circumstances, the associated sector is internally buffered.

Before calling this function, EOS calls hardware function 16 to translate the logical sector number to the physical.

Example: Set sector 2Ø for next disk access

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
            :
            :
PARAMS:   DB        11            ; Hardware function No.
          DB        Ø
          DW        2Ø            ; Sector number
          DW        Ø
          DW        Ø
```

```
**************************************************************************
*                                                                        *
*                    12 - Set DMA Address                                *
*                                                                        *
**************************************************************************
```

On entry: BC = DMA Address

On exit : ---

The parameter passed in register pair BC will be use as the starting address of the memory buffer for all future disk I/O. The buffer must have a size of at least 128 bytes inorder to accept a full data block. If multisector read/write is requested, this function is called repeatedly by EOS.

Example: Set DMA address to 80H.

```
            LD          C,50          ; EOS function code
            LD          DE,PARAMS     ; Parameters
            CALL        5             ; Call EOS
              :
              :
PARAMS:     DB          12            ; Hardware function No.
            DB          0
            DW          80H           ; DMA address
            DW          0
            DW          0
```

```
**************************************************************************
*                                                                        *
*                    13 - Read Disk Block                                *
*                                                                        *
**************************************************************************
```

On entry: ---

On exit : AL = Ø or 1

After all I/O parameters have been set by hardware functions 9, 1Ø, 11, and 12, the 128-bytes data block specified is read from disk and transferred to the memory area specified by the DMA address. This process must not necessarily cause a physical disk access, because EOS has the ability to buffer several sectors internally.

After a successful transfer, registers A and L are set to Ø. If any errors are encountered, A and L are set to 1, and an error message is displayed on the screen.

Example:

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS
          OR        A             ; Any errors ?
          JP        NZ,RDERR      ; Jump if so
            :
            :
PARAMS:   DB        13            ; Hardware function No.
          DB        Ø
          DW        Ø
          DW        Ø
          DW        Ø
```

```
**************************************************************************
*                                                                        *
*                    14 - Write Disk Block                               *
*                                                                        *
**************************************************************************
```

On entry: C = Ø, 1, or 2

On exit : AL = Ø, 1, or 2

After all I/O parameters have been set by hardware functions 9, 1Ø, 11, and 12, the 128-bytes data block is written to disk. A parameter passed in register C controls the disk write mode:

Ø = Allocated disk write. The internal sector buffer is read from disk before inserting the new block, because it contains already valid data. The buffer altered this way will not be written back to disk before it is needed for other purposes.

1 = Immediate disk write. Contrary to mode Ø, the sector buffer is immediately written back.

2 = Unallocated write. The sector buffer will not be read prior to writing, because it does not contain valid data.

After a successful write, registers A and L are returned with zero values. If a write error was encountered, both registers are set to 1. If the sector could not be written due to a write protected data carrier, A and L are 2 . In any case of error, a message is displayed on the screen.

Example: Write sector in immediate mode

```
            LD      C,5Ø          ; EOS function code
            LD      DE,PARAMS     ; Parameters
            CALL    5             ; Call EOS to execute function
            OR      A             ; was that o.k. ?
            JP      NZ,WRTERR     ; Jump if not so
              :
              :
PARAMS:     DB      14            ; Hardware function No.
            DB      Ø
            DW      1             ; Immediate buffer flush
            DW      Ø
            DW      Ø
```

```
**************************************************************************
*                                                                        *
*                    15 - Check Printer State                            *
*                                                                        *
**************************************************************************
```

On entry: ---

On exit : AL = Ø or ØFFH

If all output devices mapped via the LSTOUT: vector to the printer are ready to receive a character, A and L are set to ØFFH, otherwise Ø.

Example:

```
            LD          C,5Ø          ; EOS function code
            LD          DE,PARAMS     ; Parameters
            CALL        5             ; Call EOS to execute function
            OR          A             ; Test result
            JP          NZ,PRINT      ; Non-zero means ready
              :
              :
PARAMS:     DB          15            ; Hardware function No.
            DB          Ø
            DW          Ø
            DW          Ø
            DW          Ø
```

```
**************************************************************************
*                                                                        *
*                   16 - Translate Sector Number                         *
*                                                                        *
**************************************************************************
```

On entry: BC = Sector Number

On exit : HL = Translated Sector Number

According to the criteria specified in the DPB of the current drive, the sector number passed in register pair BC is translated into a physical block number which is returned in HL. EOS calls this function immediately before calling function 11 (set sector).

For details concerning the DPB, refer to the corresponding chapter.

Example: Translate sector number 3

```
          LD        C,50        ; EOS function code
          LD        DE,PARAMS   ; Parameters
          CALL      5           ; Call EOS to execute function
          LD        (SECNO),HL  ; Save translated sector number
            :
            :
PARAMS:   DB        16          ; Hardware function No.
          DB        0
          DW        3           ; Sector No. 3
          DW        0
          DW        0
```

```
*************************************************************************
*                                                                       *
*                  17 - Check Console Output State                      *
*                                                                       *
*************************************************************************
```

On entry: ---

On exit : AL = Ø or ØFFH

If all devices mapped to the console output channel via bit vector CONOUT: are ready to accept a character, A and L are returned with an ØFFH value, otherwise Ø.

Example:

```
            LD      C,5Ø          ; EOS function code
            LD      DE,PARAMS     ; Parameters
            CALL    5             ; Call EOS to execute function
            OR      A             ; Ready for output?
            JP      NZ,OUTYES     ; Non-zero means yes
              :
              :
PARAMS:     DB      17            ; Hardware function No.
            DB      Ø
            DW      Ø
            DW      Ø
            DW      Ø
```

```
************************************************************************
*                                                                      *
*                  18 - Check Auxiliary Device Input State             *
*                                                                      *
************************************************************************
```

On entry: ---

On exit : AL = Ø or ØFFH

All input devices mapped to the auxiliary device input channel AUXIN: are interrogated. If at least one of them is ready to be read, A and L are set to ØFFH, if not so, zero values are returned.

Example:

```
            LD          C,5Ø          ; EOS function code
            LD          DE,PARAMS     ; Parameters
            CALL        5             ; Call EOS to execute function
            OR          A             ; Check state
            JP          NZ,READY      ; non-zero means ready
              :
              :
PARAMS:     DB          18            ; Hardware function No.
            DB          Ø
            DW          Ø
            DW          Ø
            DW          Ø
```

```
*************************************************************************
*                                                                       *
*                  19 - Check Auxiliary Device Output State             *
*                                                                       *
*************************************************************************
```

On entry: ---

On exit : AL = Ø or ØFFH

If all devices mapped to the auxiliary output channel via bit vector AUXOUT: are ready to accept a character, A and L are returned with an ØFFH value, otherwise Ø.

Example:

```
            LD          C,5Ø          ; EOS function code
            LD          DE,PARAMS     ; Parameters
            CALL        5             ; Call EOS to execute function
            OR          A             ; Ready for output?
            JP          NZ,OUTYES     ; Non-zero means yes
              :
              :
PARAMS:     DB          19            ; Hardware function No.
            DB          Ø
            DW          Ø
            DW          Ø
            DW          Ø
```

```
**************************************************************************
*                                                                        *
*                  20 - Return Device Table Address                      *
*                                                                        *
**************************************************************************
```

On entry: ---

On exit : HL = Address of Device Table

The beginning address of the physical devices table is returned in HL. On the MZ-3500, this table is deposited at address 0FE00H.

Example:

```
          LD        C,50          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS to execute function
          LD        A,(HL)        ; Fetch first byte from table
            :
            :
PARAMS:   DB        20            ; Hardware function No.
          DB        0
          DW        0
          DW        0
          DW        0
```

```
**************************************************************************
*                                                                        *
*                  21 - Initialize Physical Device                       *
*                                                                        *
**************************************************************************
```

On entry: C = Device Number, Ø to 15

On exit : ---

The physical device the position of which in the device table corresponds to the number passed in register C is re-initialized. By means of this function, changes made in the table can be communicated to the corresponding device. The EOS utility $DEVICE uses this function.

Example:

```
            LD      C,5Ø          ; EOS function code
            LD      DE,PARAMS     ; Parameters
            CALL    5             ; Call EOS to execute function
              :
              :
PARAMS:     DB      21            ; Hardware function No.
            DB      Ø
            DW      2             ; Device No. 2
            DW      Ø
            DW      Ø
```

```
************************************************************************
*                                                                      *
*                    22 - Check Drive Ready                            *
*                                                                      *
************************************************************************
```

On entry: C = Drive Number, Ø to 15

On exit : H = Bit Vector
L = Ø or ØFFH

The drive specified by the number passed in register C is checked whether it is ready for I/O. A value of Ø denotes drive A, a 1 drive B and so on up to 15 for drive P.

If the drive is ready, L is set to ØFFH and H contains a bit vector as explained below:

Bit 7 = Data carrier of that drive is not removable
Bit 6 = Drive contains a two-sided floppy disk.

A zero value in L denotes that the drive is not ready; in this case, the contents of H are meaningless.

Example: See if drive B is ready

```
            LD        C,5Ø          ; EOS function code
            LD        DE,PARAMS     ; Parameters
            CALL      5             ; Call EOS to execute function
            XOR       A
            OR        L             ;
            JP        NZ,READY
              :
              :
PARAMS:     DB        22            ; Hardware function No.
            DB        Ø
            DW        'B'-'A'       ; Drive B
            DW        Ø
            DW        Ø
```

```
**************************************************************************
*                                                                        *
*                  23 - Activate Screen Editor                           *
*                                                                        *
**************************************************************************
```

On entry: C = Beginning Column
DE = Address fo Buffer

On exit : ---

The additional screen editor is activated by calling this function. In register pair DE, the address of a buffer is passed which must be set up as described at EOS function 1Ø. C denotes the first valid screen column, beginning from Ø. The system enters screen edit mode, receives an input line and deposits it in the specified buffer.

Example:

```
            LD        C,5Ø          ; EOS function code
            LD        DE,PARAMS     ; Parameters
            CALL      5             ; Call EOS to execute function
              :
              :
PARAMS:     DB        23            ; Hardware function No.
            DB        Ø
            DW        BUFFER        ; Line buffer
            DW        9             ; Begin in column 1Ø
            DW        Ø             ;   (counted from Ø)
              :
              :
BUFFER:     DB        8Ø            ; Buffer size
            DS        1             ; Number of characters typed in
            DS        8Ø            ; The buffer itself
```

```
**************************************************************************
*                                                                        *
*                    24 - Flush Sector Buffers                           *
*                                                                        *
**************************************************************************
```

On entry: ---

On exit : AL = Ø, 1, or 2

All yet unwritten sector buffers are transferred immediately out to the disks. If any errors occur, the procedure is the same as described at hardware function 14 (write block).

Example:

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS to execute function
            :
            :
PARAMS:   DB        24            ; Hardware function No.
          DB        Ø
          DW        Ø
          DW        Ø
          DW        Ø
```

```
**************************************************************************
*                                                                        *
*                    25 - Move Memory Block                              *
*                                                                        *
**************************************************************************
```

On entry: BC = Number of Bytes to be moved
DE = Source Address
HL = Destination Address

On exit : BC = Ø
DE = Source Address + Block Size
HL = Destination Address + Block Size

A memory area of <BC> bytes length is moved from address <DE> to address <HL>. This function works similar to the Z-8Ø LDIR instruction, but the usage of DE and HL is interchanged. Furthermore, only data in memory bank Ø are moved.

EOS uses this function for moving data whithin its internal working storage.

Example: Move 5ØØ bytes from 4ØØØH to 5ØØØH

```
          LD        C,5Ø          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS to execute function
            :
            :
PARAMS:   DB        25            ; Hardware function No.
          DB        Ø
          DW        5ØØ           ; Number of Bytes
          DW        4ØØØH         ; Source Address
          DW        5ØØØH         ; Destination Address
```

```
************************************************************************
*                                                                      *
*                  26 - Get / Set Calendar and Clock                   *
*                                                                      *
************************************************************************
```

On entry: C = Ø or ØFFH
DE = Time Buffer

On exit : ---

If register C is zero on entry to this function, the internal calendar and clock are read and the data are deposited in the specified buffer. If C has a value of ØFFH, date and time are transferred from the buffer into the hardware clock. The time buffer must be structured as follows:

```
            DW      DATE      ; Julian Date, where 1 = Jan 1st, 1978
            DB      15H       ; Hours, BCD format
            DB      3ØH       ; Minutes, BCD format
            DB      55H       ; Seconds, BCD format
```

Example: Set date/time to June 1st, 1983, 15:3Ø:55

```
            LD       C,5Ø          ; EOS function code
            LD       DE,PARAMS     ; Parameters
            CALL     5             ; Call EOS to execute function
              :
              :
PARAMS:     DB       26            ; Hardware function No.
            DB        Ø
            DW       ØFFH
            DW       DTETME
            DW        Ø
              :
              :
DTETME:     DW       1978          ; June 1st, 1983
            DB       15H           ; Hours
            DB       3ØH           ; Minutes
            DB       55H           ; Seconds
```

27 - Invoke Graphics

On entry: BC = Graphics Parameters

On exit : ---

This function corresponds to EOS function 115. The parameter passed there is transferred to BC. However, no check will be made of the graphics operation code. Hence, EOS function 115 should be preferred for graphic functions.

For a detailed discussion of graphic facilities, refer to section C of this manual.

Section C

Graphics Interface

(EOS Function 115)

```
**************************************************************************
*                                                                        *
*                    28 - Hexadecimal Output                             *
*                                                                        *
**************************************************************************
```

On entry: BC = Data

On exit : ---

The data passed in register pair BC is displayed on the screen as a four digit hexadecimal number.

Example: Display 10000 in hexadecimal format on the screen

```
          LD        C,50          ; EOS function code
          LD        DE,PARAMS     ; Parameters
          CALL      5             ; Call EOS to execute function
            :
            :
PARAMS:   DB        28            ; Hardware function No.
          DB        0
          DW        10000         ; Number
          DW        0
          DW        0
```

C.1. Preview

The SHARP MZ-3500 has a built-in state-of-the-art screen graphics facility. EOS offers two ways of using it: either by the Graphics Interpreter, or by EOS function 115. The latter is the subject of this section.

EOS function 115 expects a pointer to a graphic parameters block passed in DE. The first byte of the parameter block always defines the graphics instruction code, which is a number in the 0 to 24 range. Usually, the instruction code is followed by a number of further parameters which must be defined as words except for the Define Bit Image function, which requires byte parameters.

Let's have a look at an example: To set the pixel (image dot) at coordinates X = 100 and Y = 200, You would have to write:

```
          LD        C,115               ; EOS function 115
          LD        DE,GRPARM           ; Parameter area
          CALL      5                   ; Call EOS for execution
             :
             :
GRPARM:   DB        9                   ; Grahics instruction code
          DW        100                 ; X coordinate
          DW        200                 ; Y coordinate
```

C.2. Screen Definition

The graphics screen has a size of 640 pixels (image dots) wide times 400 pixels high. A pixel is specified by its coordinates X and Y, where the permitted range is 0 to 639 for X , and 0 to 399 for Y. The coordinates 0,0 denote the upper left corner.

There are two video outputs which can be operated independently. It is possible to generate two different images, because each video output can be programmed to display specific colours only. So if screen No.1 were set to display red image elements only and screen No. 2 set for green images, both screens could be referred to individually by their colour specified.

There are 3 basic colours available for graphic images. By skilful combining the basic colours, however, any of 8 colours can be achieved:

0 = black
1 = blue
2 = red
3 = magenta
4 = green
5 = cyan
6 = yellow
7 = white

C.3. Bit Image Definition

Graphics function 20 offers a facility to define one's own characters by software. You would specify a 8 x 8 bit raster which is displayed when calling the corresponding character code. These special characters are used to fill areas, too. To simplify generating line graphics, a number of these bit patterns are predefined. The definition of a new pattern could look like this:

```
Byte 1 :   00011000B  =  18H       . . . * * . . .
Byte 2 :   00100100B  =  24H       . . * . . * . .
Byte 3 :   01000010B  =  42H       . * . . . . * .
Byte 4 :   01111110B  =  7EH       . * * * * * * .
Byte 5 :   01000010B  =  42H       . * . . . . * .
Byte 6 :   01000010B  =  42H       . * . . . . * .
Byte 7 :   01000010B  =  42H       . * . . . . * .
Byte 8 :   00000000B  =  00H       . . . . . . . .
```

The pattern is defined as the character "A".

Another kind of pattern can be used to fill areas:

```
Byte 1 :   10001000B  =  88H       * . . . * . . .
Byte 2 :   01000100B  =  44H       . * . . . * . .
Byte 3 :   00100010B  =  22H       . . * . . . * .
Byte 4 :   00010001B  =  11H       . . . * . . . *
Byte 5 :   10001000B  =  88H       * . . . * . . .
Byte 6 :   01000100B  =  44H       . * . . . * . .
Byte 7 :   00100010B  =  22H       . . * . . . * .
Byte 8 :   00010001B  =  11H       . . . * . . . *
```

When executing an area-fill instruction, the area affected will be hatched.

For line graphics, hatchings, etc., bit images 0 to 12 are already predefined. They may be redefined anytime, if desired:

```
 0 = 100 percent fill
 1 = 75 percent fill
 2 = 50 percent fill
 3 = 25 percent fill
 4 = 10 percent fill
 5 = Hatching from left to right
 6 = Hatching from right to left
 7 = Cross-hatching
 8 = vertical lines
 9 = horizontal lines
10 = a "D&Z" glyph (ah, well,...)
11 = vertical serpentine lines
12 = horizontal serpentine lines
```

C.4. Table of Graphic Functions

Number	Function
Ø	Initialize Graphics
1	Erase Graphics
2	Erase Partial Graphics
3	Define Graphic Screen 1
4	Define Graphic Screen 2
5	Set Background Colour
6	Set Plot Colour
7	Set Plot Mode
8	Define Line Mask
9	Set Pixel
1Ø	Erase Pixel
11	Draw Line
12	Continue Line
13	Plot Rectangle
14	Plot Circle
15	Plot Circle Arc
16	Paint Rectangle
17	- reserved -
18	Write Character
19	Continue Writing Characters
2Ø	Define Bit Pattern
21	Hard Copy
22	Define Screen Area
23	Put Screen Area
24	Fetch Screen Area

It must be noted that EOS does not check the parameters passed. If the parameters make no sense, the results will be garbaged. In extreme cases, the system may even run out of control. So be careful not to exceed the limits!

```
**************************************************************************
*                                                                        *
*                    Ø - Initialize Graphics                             *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    Ø          ; Graphics Instruction Code
```

The Graphics driver is re-initialized.

```
**************************************************************************
*                                                                        *
*                    1 - Erase Graphics                                  *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    1          ; Graphics Instruction Code
```

All Graphics are erased.

```
**************************************************************************
*                                                                        *
*                    2 - Erase Partial Graphics                          *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    2          ; Graphics Instruction Code
             DW    X1,Y1      ; Upper left coordinates
             DW    X2,Y2      ; Lower right coordinates
             DW    COLOUR     ; Colour
```

This function allows for selective erasure of graphics. Within the rectangle specified by its diagonal coordinates X1,Y1 and X2,Y2, the colour combination COLOUR is erased in all objects. COLOUR is regarded as a combination of three bits meaning:

Bit Ø = blue parts, bit 1 = red parts, bit 2 = green parts.

The values permitted for the parameters are as follows:

X - Ø to 639
Y - Ø to 399
COLOUR - Ø to 7

```
**************************************************************************
*                                                                        *
*                    3 - Define Graphic Screen 1                         *
*                                                                        *
**************************************************************************

Parameters:  DB    3          ; Graphics Instruction Code
             DW    BITS       ; Bit Vector
```

Graphic screen 1 is set up for graphic output. The bit vector specifies:

Bit 0 : Display blue parts of graphics.
Bit 1 : Display red parts of graphics.
Bit 2 : Display green parts of graphics.
Bit 3 : Set for colour monitor, reset for black/white monitor.
Bit 4 : Set to mix graphics and normal text on screen.

```
**************************************************************************
*                                                                        *
*                    4 - Define Graphic Screen 2                         *
*                                                                        *
**************************************************************************

Parameters:  DB    4          ; Graphics Instruction Code
             DW    BITS       ; Bit Vector
```

Graphic screen 2 is set up for graphic output. The bit vector specifies:

Bit 0 : Display blue parts of graphics.
Bit 1 : Display red parts of graphics.
Bit 2 : Display green parts of graphics.
Bit 3 : Set for colour monitor, reset for black/white monitor.
Bit 4 : Set to mix graphics and normal text on screen.
Bit 5 : Set to display character attributes, as there are blinking, inverse video, etc.

```
************************************************************************
*                                                                      *
*                    5 - Set Background Colour                         *
*                                                                      *
************************************************************************
```

```
Parameters:  DB     5          ; Graphics Instruction Code
             DW     COLOUR     ; Colour
```

The background colour is specified. The COLOUR value must be in the Ø to 7 range. The background colour is independent of graphics, it may be active even with normal screen use, if the screen is operated in colour mode.

```
************************************************************************
*                                                                      *
*                     6 - Set Plot Colour                              *
*                                                                      *
************************************************************************
```

```
Parameters:  DB     6          ; Graphics Instruction Code
             DW     COLOUR     ; Colour
```

All subsequent graphic objects will be plotted in the colour specified, which must be coded as a number between Ø and 7.

```
**************************************************************************
*                                                                        *
*                      7 - Set Plot Mode                                 *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     7          ; Graphics Instruction Code
             DW     MODE       ; Mode
```

The plot mode is defined. Four different plot modes are available:

```
     0 = Overwrite
     1 = Invert colour
     2 = Reset colour (effectively, erase)
     3 = Additive mixing of colours
```

```
**************************************************************************
*                                                                        *
*                      8 - Define Line Mode                              *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     8          ; Graphics Instruction Code
             DW     MASK       ; Bit Vector
```

The line mode for subsequent plot instructions is defined. Lines or "vectors" are composed of a sequence of pixels (dots) which is defined by this function. For instance, a dashed line may be defined by the bit pattern

1100110011001100 (binary)

```
**************************************************************************
*                                                                        *
*                       9 - Set Pixel                                    *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     9           ; Graphics Instruction Code
             DW     X,Y         ; Coordinates
```

The pixel at coordinates X,Y is set, i.e. lit up in the predefined plot colour. The coordinates may assume the following values:

```
     X       - Ø to 639
     Y       - Ø to 399
```

```
**************************************************************************
*                                                                        *
*                      1Ø - Erase Pixel                                  *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     1Ø          ; Graphics Instruction Code
             DW     X,Y         ; Coordinates
```

The pixel at coordinates X,Y is reset. The coordinates may assume the following values:

```
     X       - Ø to 639
     Y       - Ø to 399
```

```
**************************************************************************
*                                                                        *
*                      11 - Draw Line                                    *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    11          ; Graphics Instruction Code
             DW    X1,Y1       ; Beginning coordinates
             DW    X2,Y2       ; Final coordinates
```

A straight line is drawn from the point at coordinates X1, Y1 to the point at coordinates X2, Y2. The line colour is determined by graphic function 6, the line mask by function 8. The coordinates are limited to the following intervals:

```
     X       - Ø to 639
     Y       - Ø to 399
```

```
**************************************************************************
*                                                                        *
*                      12 - Continue Line                                *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    12          ; Graphics Instruction Code
             DW    X,Y         ; Final coordinates
```

A straight line is drawn from current position of the (invisible) graphic cursor to the point at coordinates X2, Y2. The line colour is determined by graphic function 6, the line mask by function 8. The coordinates must lie within following intervals:

```
     X       - Ø to 639
     Y       - Ø to 399
```

By means of this function, polygonials can be drawn.

```
**************************************************************************
*                                                                        *
*                      13 - Plot Rectangle                               *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    13          ; Graphics Instruction Code
             DW    X1,Y1       ; Upper left corner
             DW    X2,Y2       ; lower right corner
```

Both coordinate pairs denote the upper left resp. the lower right corners of a rectangle to be plotted. The line colour is determined by graphic function 6, the line mask by function 8. The coordinates must lie within following intervals:

```
X       - 0 to 639
Y       - 0 to 399
```

```
**************************************************************************
*                                                                        *
*                      14 - Plot Circle                                  *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    14          ; Graphics Instruction Code
             DW    X,Y         ; Centre of Circle
             DW    R           ; Radius
```

A circle of radius R is drawn around the centre specified by its coordinates X,Y. The line colour is determined by graphic function 6, the line mask by function 8. The coordinates must lie within the following intervals:

```
X       - 0 to 639
Y       - 0 to 399
```

It should be observed that the complete circle will fit on the screen.

```
************************************************************************
*                                                                      *
*                    15 - Plot Arc of Circle                           *
*                                                                      *
************************************************************************
```

```
Parameters:  DB     15          ; Graphics Instruction Code
             DW     X,Y         ; Centre
             DW     R           ; Radius
             DW     PHI1        ; Starting Angle
             DW     PHI2        ; Final Angle
```

A circle arc of radius R is plotted, centered at coordinates X,Y. The starting and final angles are to be specified in degrees, where Ø degree represents the rightmost point of a circle on the screen. The line colour is determined by graphic function 6, the line mask by function 8. The coordinates must lie within the following intervals:

```
     X       - Ø to 639
     Y       - Ø to 399
     PHI     - Ø to 359
```

```
************************************************************************
*                                                                      *
*                    16 - Fill Rectangle                               *
*                                                                      *
************************************************************************
```

```
Parameters:  DB     16          ; Graphics Instruction Code
             DW     X1,Y1       ; Upper left corner
             DW     X2,Y2       ; Lower right corner
             DW     COLOUR      ; Colour
             DW     IMAGE       ; Image Number
```

The coordinates X1, Y1 and X2,Y2 denote the diagonal of a rectangle which is filled with a certain pattern. The pattern is a multiple of one of the 8x8-bit images which can be defined by graphic function 2Ø. COLOUR specifies the colour of the bit pattern. Following values are permitted:

```
     X       - Ø to 639
     Y       - Ø to 399
     COLOUR  - Ø to 7
     IMAGE   - Ø to 126
```

```
************************************************************************
*                                                                      *
*                    18 - Write Character                              *
*                                                                      *
************************************************************************
```

```
Parameters:  DB    18          ; Graphics Instruction Code
             DW    X,Y         ; Starting Coordinates
             DW    CHAR        ; Character
             DW    DIRECT      ; Character Writing Direction
             DW    FACTOR      ; Magnification Factor
```

The character CHAR is written as a graphic pattern. The coordinates X,Y specify the lower left corner of the character pattern area. The writing direction is denoted by DIRECT. The character may be magnified, where a FACTOR of 0 means a 1:1 scale, 1 a 1:2 scale, 2 a 1:3 scale, and so forth. The character itself is specified by its equivalent ASCII code. All images may be altered by graphic function 20.

The value DIRECT and the writing direction correspond as follows:

```
0 = Normal (from left to right)
1 = 45 degrees inclining to upper right
2 = Vertically up, 90 degrees
3 = 135 degrees
4 = Horizontally, but head down to left (180 degrees)
5 = 45 degrees declining to lower left, head down (225 degrees)
6 = Vertically downwards (270 degrees)
7 = 45 degrees declining to lower right (315 degrees)
```

The parameters must obey to the following bounds:

```
X       - 0 to 639
Y       - 0 to 399
CHAR    - 0 to 126
DIRECT  - 0 to 7
FACTOR  - 0 to 15
```

A value of 10 to 17 for DIRECT is giving the same writing directions as mentioned above but the characters will be slanted.

```
**************************************************************************
*                                                                        *
*                    19 - Continue Writing Characters                    *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     19          ; Graphics Instruction Code
             DW     CHAR        ; Character
```

At the current position of the (invisible) graphic cursor, a character is written. The character is formatted as defined by graphic function 18.

This function must be called only after function 18 is completed. Its purpose is to write complete strings of characters.

```
**************************************************************************
*                                                                        *
*                    2Ø - Define Bit Image                               *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB    2Ø          ; Graphics Instruction Code
             DW    CHAR        ; Number of Character
             DB    BITS1       ; Bit Pattern 1
                :
                :
             DB    BITS8       ; Bit Pattern 8
```

This function provides for a means for defining characters by software. An image of 8 time 8 bit is defined and stored at the position CHAR, which can be referred to by functions 18 and 19. Furthermore, the pattern can be used as filling pattern along with function 16. For CHAR, values between Ø and 126 are permitted.

Bit images Ø to 12 are predefined with fill patterns in order to simplify generating block graphics.

```
**************************************************************************
*                                                                        *
*                      21 - Hard Copy                                    *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     21          ; Graphics Instruction Code
             DW     COLOUR      ; Colour
```

The graphic screen is transferred to the SHARP Ink-Jet printer IO-700 to obtain a hard copy. The bit vector COLOUR specifies which partial colours are to be printed:

```
     Bit 0 = Print blue parts of graphics
     Bit 1 = Print red parts of graphcis
     Bit 2 = Print green parts of graphics
```

It is thus possible to generate a hard copy with specific colours.

```
**************************************************************************
*                                                                        *
*                      22 - Define Screen Area                           *
*                                                                        *
**************************************************************************
```

```
Parameters:  DB     22          ; Graphics Instruction Code
             DW     X1,Y1       ; Upper left corner
             DW     X2,Y2       ; Lower right corner
             DW     COLOUR      ; Colour
```

A screen area is defined, which may subsequently be written to or read from a file. X1,Y1 and X2,Y2, denote the upper left and lower right corner of a rectangular area of the screen following function calls will refer to. COLOUR is a bit vector denoting:

```
     Bit 0 = include blue parts
     Bit 1 = include red parts
     Bit 2 = include green parts
```

```
************************************************************************
*                                                                      *
*                    23 - Put Screen Area                              *
*                                                                      *
************************************************************************
```

```
Parameters:  DB    23          ; Graphics Instruction Code
```

This function may be called only after a call of graphic function 22. A graphic data block of 128 bytes is read from the current DMA buffer and put to the graphic screen. If HL is Ø upon return from this function, the transfer of data to the graphic screen is completed, otherwise, further data blocks are expected.

No other calls of EOS function 115 are allowed between a call of graphic function 22 and repeated calls of graphic funtion 23. If this rule is not observed, transfer will be aborted.

```
************************************************************************
*                                                                      *
*                    24 - Fetch Screen Area                            *
*                                                                      *
************************************************************************
```

```
Parameters:  DB    24          ; Graphics Instruction Code
```

By means of this function, a graphic data block of 128 is fetched from the graphic screen and deposited at the current DMA buffer. If HL is Ø upon return from this function, the readout of data from the graphic screen is completed, otherwise, further data blocks are to be transferred.

No other calls of EOS function 115 are allowed between a call of graphic function 22 and repeated calls of graphic function 24. If this rule is not observed, transfer will be aborted.

Section D

CRT Interface

D.1. Preview

In this section, the CRT interface is described. This interface "understands" a number of control codes, nearly all of them beginning with ASCII control character "ESCAPE" (1B hex, 27 decimal), followed by further characters. For example, when in the subsequent paragraphs, a character string "ESC A" is referred to, this is to be understood as sending the control character "ESC", followed by the character "A".

D.2. The Standards Implemented

In order to make the application of existing software as easy as possible, the MZ-3500 processes the control codes of two widely used standard terminals. There are two control codes to switch between both terminals:

ESC 'A' = TeleVideo 950 Terminal (TV950)

ESC 'B' = Lear Siegler ADM3A Terminal, as implemented in the MZ-80B CP/M version.

After cold start and program termination, TeleVideo mode is selected by default.

All system programs are adapted to TeleVideo mode. If an application program is to run under ADM3A mode, this mode must be selected everytime when needed (e.g. in the program start message). When setting and resetting the ADM3A mode, the screen is erased. We recommend that all programs should be adapted to the substantially more comfortable TeleVideo mode.

Specifications:	TV950	ADM3A
Lines per screen	24	25
Characters per line	80	80
Status line =>	yes (line 25)	no

D.3. Table of Control Codes implemented

Deviations from the standard are marked by an asterisk (*). Further explanations (=>) are given later in the text.

The lead-in code is predefined as <ESC> (1BH).

All references to <x> and <y> are to be understood as <x+20h> and <y+20h>, i.e <space> denotes position 0.

Cursor functions:	TV950	ADM3A
Line feed	Ctrl-J	Ctrl-J / ESC '*'
Carriage return	Ctrl-M	Ctrl-M / ESC '-'
New Line (CR+LF)	-----	Ctrl-T / ESC '4'
Tabulator	Ctrl-I	Ctrl-I
Cursor home (0,0)	Ctrl-^	Ctrl-^ / ESC '>'
Cursor up	Ctrl-K	Ctrl-K / ESC '+'
Cursor down	Ctrl-V	* Ctrl-V / ESC '6'
Cursor left	Ctrl-H	Ctrl-H / ESC '('
Cursor right	Ctrl-L	Ctrl-L / ESC '''
Set cursor position =>	ESC '=' <y> <x>	Ctrl-] / ESC '='
Read cursor position =>	ESC '?'	Ctrl-\ / ESC '<'

Erase functions:	TV950	ADM3A
Entire screen	Ctrl-Z / ESC '*'	Ctrl-Z / ESC ':'
Cursor to end-of-screen	ESC 'Y' / ESC 'y'	Ctrl-S / ESC '3'
Cursor to end-of-line	ESC 'T' / ESC 't'	Ctrl-R / ESC '2'
Cursor to page position	* ESC '6' <y> <x>	-----
Cursor to line position	* ESC '5' <x>	-----

Insert and Delete:	TV950	ADM3A
Insert character	ESC 'Q'	Ctrl-N / ESC '.'
Insert line	ESC 'E'	Ctrl-O / ESC '/'
Erase character	ESC 'W'	Ctrl-P / ESC '0'
Erase line	ESC 'R'	Ctrl-Q / ESC '1'

Attributes:	TV950	ADM3A
Inverse single characters	ESC ')'	Ctrl-C / ESC '#'
Normal singele characters	ESC '('	Ctrl-D / ESC '$'
Character attributes =>	ESC 'G' <n>	* Ctrl-E / ESC '%'
Cursor Attributes =>	ESC '.' <n>	* Ctrl-F / ESC '&'

Screen modes:	TV950	ADM3A
Entire screen inverse	ESC 'b'	Ctrl-A / ESC '!'
Entire screen normal	ESC 'd'	Ctrl-B / ESC '"'
Video on	* ESC 'l' =>	* Ctrl-U / ESC '5'
Video off	* ESC 'ø' =>	* Ctrl-U / ESC '5'

Special functions:	TV950	ADM3A
Bell (Beep)	Ctrl-G	Ctrl-G / ESC '''
Initialize TV950 mode	* ESC 'A'	* ESC 'A'
Initialize ADM3A mode	* ESC 'B'	* ESC 'B'
Start graphics interpreter	* ESC 'I'	* ESC 'I'
Enter mask graphics mode =>	ESC '$'	-----
End mask graphics mode =>	ESC '%'	-----
Init. keyboard and screen	* ESC '['	ESC '['
Set CRT colour mode =>	* ESC 'C' <n>	* ESC 'C' <n>
Set CRT extra characterset	* ESC '3' <n>	* ESC 'D' <n>
Set keyboard "Caps lock" =>	* ESC '4' <n>	* ESC 'E' <n>
Activate screen editor =>	* Ctrl-Y / ESC 'N'	* Ctrl-Y / ESC '9'
Toggle Monitor mode =>	* Ctrl-underline	* Ctrl-underline
Screen hardcpoy =>	* ESC 'P'	* Ctrl-W / ESC '7'
Redefine keys =>	* ESC '|' <....>	* Ctrl-X / ESC '8'
Read key definitions =>	* ESC '@'	* ESC '@'
Lock keyboard	ESC '#'	* ESC 'F'
Release keyboard	ESC '"'	* ESC 'G'
Clock display on/off =>	* ESC '2' <n>	-----
Window scrolling =>	* ESC '7' <xl..y2>	* ESC 'H' <xl..y2>

Status line (TV950 only):	TV950
Display system status line	ESC 'h'
Display user status line	ESC 'g'
Set system status line	* ESC 'e' <n> <Text> <CR> =>
Set user status line	* ESC 'f' (ESC 'G' <att>) <Text> <CR>

D.4. Detailed Description of Control Sequences

D.4.1. Set Cursor Position

TV950: <ESC> <'='> <y+20h> <x+20h>
y = 0...23 / x = 0...79

ADM3A: <Ctrl-]> <y+20h> <x+20h> or <ESC> <'='> <y+20h> <x+20h>
y = 0...24 / x = 0...79

The position coordinates y (line) and x (column) must be specified with an offset of 20H, respectively.

Example: <ESC> <'='> <space> <space> sets the cursor to its "home" position (0,0).

D.4.2. Read Cursor Position

TV950: <ESC> <'?'>

ADM3A: <Ctrl-\> or <ESC> <'<'>

The cursor position is returned in the same form of key codes as it is set. The transfer of the position is terminated by <CR>.

Example: The cursor is set to position y = 5 and x = 4.
Following codes will be returned:
<'%'> <'$'> <CR>

D.4.3. Set Character Attributes

A set attribute is valid until another is set, or the screen is re-initialized.

TV950: <ESC> <'G'> <n>

ADM3A: <Ctrl-E> <n> or <ESC> <'%'> <n>

Only the least significant 4 bits of <n> are evaluated, their meanings are:

```
Bit 0  =  Vertical line right of character
    1  =  blinking characters
    2  =  inverse characters
    3  =  characters underlined
```

```
Example:  <ESC> <'G'> <'0'>  =  No attributes
          <ESC> <'G'> <'4'>  =  Inverse characters
          <ESC> <'G'> <'<'>  =  Blinking inverse characters
```

It must be noted that the TeleVideo terminal treats attributes in a different manner. Setting an attribute on the TV950 causes a space being written on the screen at the cursor position, and the remainder of the screen is displayed using the attribute until another attribute is encountered or up to the end of the screen. Due to organizational reasons, only the characters written after setting an attribute are displayed with the attribute, instead of the entire remaining screen. Additionally, no space is written with the MZ-3500 implementation.

D.4.4. Set Cursor Attributes

A set attribute is valid until another is set, or the screen is re-initialized.

After a warm start or a CRT re-initialization, the cursor will appear as a steady block. By means of the following sequence, it can be changed in various ways:

TV95Ø: <ESC> <'.'> <n>

ADM3A: <Ctrl-F> <n> or <ESC> <'&'> <n>

<n> may assume (ASCII) values as follows:

Ø = Cursor off (not displayed)
1 = Cursor is blinking block
2 = Cursor is steady block
3 = Cursor is blinking underline
4 = Cursor is steady underline

Example: <ESC> <'.'> <'1'> causes the cursor to be displayed as a blinking block.

D.4.5. Video On / Off

It is possible to suppress the screen display. All CRT functions remain fully active, but the screen is blanked until video is switched on again.

TV95Ø: Video on = <ESC> <'1'>
Video off = <ESC> <'Ø'>

ADM3A: Video on = <Ctrl-U> <'1'> or <ESC> <'5'> <'1'>
Video off = <Ctrl-U> <'Ø'> or <ESC> <'5'> <'Ø'>

D.4.6. Activate Screen Editor

The screen editor enables to edit on the entire screen area. All cursor and other control functions can be used. Edit mode is terminated by typing <CR>. This effects that the complete line in which the cursor is currently standing, will be returned as an input from the keyboard, terminated by <CR>. The following screen area is erased so that the screen output from the function called is clearly readable.

TV950: <Ctrl-Y> <n+20H> or <ESC> <'N'> <n+20H>

ADM3A: <Ctrl-Y> <n+20H> or <ESC> <'9'> <n+20H>

n denotes the number of characters at the beginning of the line which are to be ignored, e.g. a previous output.

This function greatly enhances the operating comfort, especially with the EOS "Line Input" function.

D.4.7. Monitor Mode

By <Ctrl-underline> the Monitor mode is toggled between four different modes which are effectiv especially in the "Edit Mode" when testing software:

Monitor Mode disabled	-	all control characters from input and output are processed in the normal way.
full Monitor Mode	-	all control characters from input and output are not processed, but displayed as tilted characters.
CRT Monitor Mode	-	only control characters from keyboard are processed, outputs are monitored.
KBD Monitor Mode	-	only control characters send to the screen are processed, inputs from keyboard are monitored.

D.4.8. Screen Hardcopy

The screen contents are transferred directly to the printer connected to the Centronics interface. In TV950 mode, the status line is not printed.

TV950: <ESC> <'P'>

ADM3A: Ctrl-W / <ESC> <'7'>

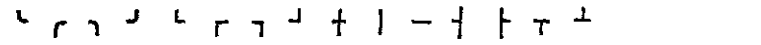
D.4.9. Mask Graphics Mode

The TeleVideo implementation gives you the possibility to draw masks of lines on the screen. To enter this special mode use

TV95Ø: <ESC> <'$'> and <ESC> <'%'> to exit it.

If this mode is active, any alpha-numeric character 'A' to 'P' resp. 'a' to 'p' is converted to one of 16 pseudo graphic characters:

A B C D E F G H I J K L M N O P



D.4.1Ø. Special Screen and Keyboard Modes

CRT colour mode	TV95Ø:	ESC 'C' <n>	ADM3A:	ESC 'C' <n>
CRT extra characterset		ESC '3' <n>		ESC 'D' <n>
keyboard "Caps lock"		ESC '4' <n>		ESC 'E' <n>

Where <n> = 'Ø' - Mode is switched off and '1' - Mode is switched on.

D.4.11. Clock Display On/Off

In TeleVideo mode, the current date and time can be displayed in the status line. Several modes are available:

TV95Ø: <ESC> <'@'> <n> where <n> = 'Ø' - Clock-"on"
= '1' - Clock-"off"
> '1' - Clock-"off" is locked

If the parameter specified is 'Ø' or '1', the clock display can be switched on/off by => control function <Ctrl-3>.

D.4.12. Window Scrolling

TV95Ø: <ESC> <'7'> <Y1+2Øh> <X1+2Øh> <Y2+2Øh> <X2+2Øh>
ADM3A: <ESC> <'H'> <Y1+2Øh> <X1+2Øh> <Y2+2Øh> <X2+2Øh>

With every call the window with edges X1,Y1 and X2,Y2 is moved by one line in vertical direction. It is moved up if Y1 is bigger than Y2 otherwise down. The first line (Y1) is always lost and the last line (Y2) is blanked afterwards.

D.5. The TV950 Mode Status Line

In TV950 mode, line 25 is predefined as information line. It is separated from the regular screen functions. In this line, either the system status or a free definable user status can be displayed. Switching from system status to user status can be done arbitrarily, both lines being completely independent from one another.

D.5.1 The System Status Line

TV950: <ESC> <'h'> - Display system status line

The system status line (displayed in inverse mode) contains 8 fields with informations on the current status of EOS:

Field '0' - Name of current program.
(the one most recently started by the command interpreter)

Field '1' - Default drive.

Field '2' - Active user area.

Field '3' - Keyboard and screen state:

```
' KeyLock '  -  Keyboard is locked, no input possible
' Monitor '  -  Screen in => Monitor mode
' Key Def '  -  Input of => Key definitions
' UsrStat '  -  Setting => User status line
' SysStat '  -  Setting => System status line
'Graphics '  -  Screen in => Pseudo graphics mode
'Edit Mode'  -  specific => Edit mode active
'Hard Copy'  -  => Screen Hardcopy is just performed
```

Field '4' - When EOS executes a command file, the message "Execute" appears here.

Field '5' - free for user programs.

Field '6' - Date (can be switched off)

Field '7' - Time (can be switched off)

All fields of the system status line may be written to by user programs with messages of their own:

TV950: <ESC> <'e'> <n> <..text..> <CR>

<n> = No. of field ('0'..'7' / greater = system message)
<..text..> = up to 9 characters
<CR> = end of message

As long as the clock display is active, user programs should avoid writing to fields '6' and '7'. If number <'8'> is specified for <n>, the original system status line is re-activated.

Example: <ESC> <'e'> <'8'> <CR> = generate system status line by EOS and subsequently, display by <ESC> <'h'>.

D.5.2. The User Status Line

TV950: <ESC> <'g'> - Display user status line.

TV950: <ESC> <'f'> (<ESC> <'G'> <a>) <..text..> <CR> - set up

<a> = attributes as for screen (e.g. '2' = blink), but valid for entire line.

<..text..> = up to 80 characters of text

<CR> = end of text

Setting an attribute is optional. If the character string (<ESC> <'G'> <a>) is not sent, inverse will be used as attribute by default.

D.5.3. Suppressing the Status Line

By sending the character string

<ESC> <'f'> <CR> <ESC> <'g'>,

the entire status line can be suppressed.

D.6. The Keyboard

The keyboard is devided into three functional blocks:

1. Alpha-numerical full keyboard with some function keys,
2. Numerical (decimal) keypad,
3. free definable function keys.

The keyboard can be locked by means of the <PRO>-<OP> switch. The current keyboard state is indicated in field No. 2 of the system status line.

In <PRO> (program) setting, the keyboard is released.
In <OP> (operation) setting, the keyboard is locked.

D.6.1. The Alpha-numerical Full Keyboard

The keyboard corresponds to the QWERTY standard. The function keys placed here (TAB and cursor control) are, like most other function keys, free definable. The only non-redefinable keys are:

<ENTER> = <CR> = ØDh
<CMD> = <ESC> = 1Bh

D.6.2. The Numerical Keypad

In this block, the keys <ØØ> and <.> are free definable.

By pressing one of the keys <Ø> to <9> simultaneously with the <CTRL> key, the associated "control function" (represented by <Ctrl-Ø> to <Ctrl-9>) is invoked. The available "control functions" are:

<Ctrl-Ø> = The keyboard input buffer, which can take up a maximum of 14Ø characters, is erased completely.

<Ctrl-1> = The effect of the keys <SHIFT> and <LOCK> is interchanged, thus providing for an independent change between uppercase and lowercase mode.

<Ctrl-3> = Switch clock display in status line on / off.

<Ctrl-4> = Perform a hardcopy of the screen contents onto a printer via the parallel interface.

<Ctrl-5> = Perform a hardcopy of the graphics on the SHARP ink jet printer IO-7ØØ.

<Ctrl-6> = Switch graphics display on / off.

<Ctrl-7> = Start of manual key definition.

All input is echoed on the screen. First, the key to be defined must be pressed, which is then displayed in plain text. If <ESC> (=<CMD>) is pressed, the original system definition is re-installed. Up to 128 characters of text can be associated to one single key. The key definition memory can take up a maximum of 3ØØ characters.

If during entering the definition, <Ctrl-7> is pressed, the definition process is aborted.

<Ctrl-8> = Terminate manual key definition. The new text is then associated to the specified key.

<Ctrl-9> = Re-initialization of screen and keyboard.

D.6.3. Free Definable Function Keys

Except of <CMD> and <ENTER>, each single function key can be associated with arbitrary character strings by the user or his programs. This is achieved either manually by => "control functions" <Ctrl-7> and <Ctrl-8>, or by control sequences the program sends to the screen driver.

TV950: <ESC> <'|'> <n> <.text.> <Ctrl-Y> .... <Ctrl-Y> <Ctrl-Y>

ADM3A: <Ctrl-X> <n> <.text.> .... <n> <.text.> <Ctrl-Y> <Ctrl-Y>

<n> = Code of the key to be redefined.

<Text> = up to 128 characters of text per key. All control codes are allowed. The codes <Ctrl-P> and <Ctrl-Y> must be lead in by another <Ctrl-P>.

By one single system call, several keys may be defined in sequence. Each definition must be terminated by <Ctrl-Y>. The entire definition sequence must be terminated by two <Ctrl-Y> immediately following one another. Up to 300 characters can be stored for all keys together.

D.6.3.1. Key Definition Control Codes

```
30H  =  '0'  :  Clear all definitions.
31H  =  '1'  :  Store current definition.
32H  =  '2'  :  Re-activate stored definition.
33H  =  '3'  :  Re-install system definition.
```

The table below shows in which manner the function keys are "addressed" by ASCII characters during the key redefinition process.

ASCII Codes	Keys		ASCII Codes	Keys
41H = 'A' :	<F1>		60h = '`' :	<Ctrl-CMD>
42H = 'B' :	<F2>		61h = 'a' :	<Ctrl-F1>
43H = 'C' :	<F3>		62h = 'b' :	<Ctrl-F2>
44H = 'D' :	<F4>		63h = 'c' :	<Ctrl-F3>
45H = 'E' :	<F5>		64h = 'd' :	<Ctrl-F4>
46H = 'F' :	<F6>		65h = 'e' :	<Ctrl-F5>
47H = 'G' :	<F7>		66h = 'f' :	<Ctrl-F6>
48H = 'H' :	<F8>		67h = 'g' :	<Ctrl-F7>
49H = 'I' :	<F9>		68h = 'h' :	<Ctrl-F8>
4AH = 'J' :	<F10>		69h = 'i' :	<Ctrl-F9>
			6Ah = 'j' :	<Ctrl-F10>
4BH = 'K' :	<HOME>		6Bh = 'k' :	<Ctrl-HOME>
4CH = 'L' :	<DEL>		6Ch = 'l' :	<UP>
4DH = 'M' :	<INS>		6Dh = 'm' :	<DOWN>
4Eh = 'N' :	<-ENTER>		6Eh = 'n' :	<RIGHT>
4Fh = 'O' :	<CL>		6Fh = 'o' :	<LEFT>
50h = 'P' :	<RUN>			
51h = 'Q' :	<EDIT>			
52h = 'R' :	<DEB>			
53h = 'S' :	<Ctrl-DEB>			
54h = 'T' :	<BRK>			
55h = 'U' :	<Ctrl-BRK>			
56h = 'V' :	<00>			
57h = 'W' :	<.> in numerical keypad			
58h = 'X' :	<TAB>			

6.4. Function Key Predefinitions

Most function keys are predefined by the system:

Key	Definition	Function
fixed definition:		
<CMD>	1Bh = <ESC>	Lead-in for screen control codes
<ENTER>	ØDh = <CR>	Pass line to EOS.
re-programmable:		
<TAB>	Ø9h = <TAB>	Horizontaler tabulator
<INS>	ØFh = <Ctrl-O>	Insert character
<DEL>	7Fh = <RUBOUT>	Delete character
<DEB>	1Fh = <Ctrl-Underln>	Monitor mode on/off
<BRK>	13h = <Ctrl-S>	Suspend screen output
<Ctrl-BRK>	Ø3h = <Ctrl-C>	Abort program
<UP>	ØBh = <Ctrl-K>	Cursor one line up
<DOWN>	16h = <Ctrl-V>	Cursor one line down
<LEFT>	Ø8h = <Ctrl-H>	Cursor one character to the left
<RIGHT>	ØCh = <Ctrl-L>	Cursor one character to the right
<HOME>	1Eh = <Ctrl-^>	Cursor to Ø,Ø (home position)
<Ctrl-HOME>	1Ah = <Ctrl-Z>	Clear screen
<CL>	18h = <Ctrl-X>	Erase input line
<-ENTER>	<ESC> <'T'>	Erase to end-of-line
<ØØ>	<'Ø'> <'Ø'>	Double Zero
<.>	<'.'>	Decimal point

D.6.4. Key Programming Example

For a word processor, the key definition subprogram could look like this:

```
EOS     EQU     5               ; EOS system entry address
ESC     EQU     27              ; ASCII for <ESC>
CTLY    EQU     25              ; <Ctrl-Y> = separator
CTLP    EQU     16              ; <Ctrl-P> = lead-in character

        ORG     100H            ; Start by EOS
        JP      INIT            ; init keydefs before start
        JP      RESET           ; reset keydefs before end

        ORG     MEMORY          ; free memory

INIT:   LD      DE,INITBL       ; Table with new definitions
        CALL    SETDEF          ; redefine keys
        JP      START           ; normal starting address

RESET:  LD      DE,RESTBL       ; Code for re-initialization
        CALL    SETDEF          ; reset key definitions
        JMP     0               ; and back to EOS

SETDEF: PUSH    DE              ; save address of table
        LD      C,109           ; EOS: Console Mode
        LD      DE,0FFFFH       ; old value
        CALL    EOS             ;  ..read
        LD      (OLDMOD),HL     ;  .. and save
        LD      C,109           ; EOS: Console Mode
        LD      DE,0FH          ; -no transformations-
        CALL    EOS             ; set
        POP     DE              ; fetch address again
        LD      C,111           ; EOS: Print block
        CALL    EOS             ; tabl to screen driver
        LD      DE,(OLDMOD)     ; old Console Mode
        LD      C,109           ; EOS: Console Mode
        CALL    EOS             ;    reset
        RET
```

```
OLDMOD: DW      Ø                       ; space for old Console Mode

INITBL: DW      INIMSG,INILEN           ; address and length
INIMSG: DB      ESC,'|'                 ; start keydef
        DB      '1',CTLY                ; save old definitions.
        DB      'Ø',CTLY                ; clear all
        DB      'o',8,CTLY              ; <-  = ^H
        DB      'n',4,CTLY              ; ->  = ^D
        DB      'l',5,CTLY              ; UP  = ^E
        DB      'm',24,CTLY             ; DOWN  = ^X
        DB      'K',ESC,'BV',13,CTLY    ; HOME = top screen
        DB      'k',ESC,'-BV',13,CTLY   ; ^HOME = bottom screen
        DB      'O',CTLP,CTLY,CTLY      ; CL = ^Y
        DB      'L',127,CTLY            ; DEL = RUBOUT
        DB      'M',15,CTLY             ; INS = ^O
        DB      'N',14,CTLY             ; -ENTER = insert line
        DB      'X',9,CTLY              ; TAB
        DB      'V','ØØ',CTLY           ; ØØ
        DB      'W','.',CTLY            ; DECIMAL POINT
        DB      CTLY                    ; End of table
INILEN  EQU     $-INIMSG

RESTBL: DW      RESMSG,RESLEN
RESMSG: DB      ESC,'|'
        DB      '2',CTLY,CTLY           ; re-insert old definition
RESLEN  EQU     $-RESMSG

        END
```