ZSDOS 1.1

A replacement of the CP/M 2.2 BDOS

Programmer's Manual

by

Harold F. Bower
Cameron W. Cotrill
Carson Wilson

Translation from German by

Wayne Hortensius

Word/PDF conversion by

Randy Merkel (2025)

Table of Contents

1 Introduction

ZSDOS is a powerful replacement for the Basic Disk Operating System (BDOS) of CP/M 2.2 or ZRDOS 1.x systems. It has various new and numerous improved functions, but the compatibility with existing CP/M 2.2 programs is largely preserved. Maintaining this compatibility was also the main goal when developing ZSDOS. In some cases, however, the possibilities of integrating elements from CP/M Plus (also known as CP/M 3) and from ZRDOS seemed more important to us than maintaining compatibility. Furthermore, expandable data structures were designed and integrated in ZSDOS to create the basis for a more powerful operating system.

This manual describes in detail the functions, interfaces and data structures of ZSDOS. In particular, the focus is on the new features of ZSDOS, but also the less known or often misunderstood properties of CP/M. Short examples in Z80 assembly language are intended to illustrate the use of the ZSDOS function calls. In the appendices you will find short overviews of the functions of ZSDOS.

This programmer's manual was not designed as complete documentation of the CP/M operating system or the Z80 assembly language. A certain knowledge of the conventions when using CP/M function calls is required. For more comprehensive information on CP/M and the Z80 assembly language, we recommend the works listed in the bibliography of the ZSDOS User's Guide. Unless a special DOS is indicated, the statements made in this manual about ZSDOS also apply to ZDDOS.

1.1 Operating system components

The operating system CP/M 2.2 (or compatible) consists of three separate segments; the Basic Input/Output System (BIOS), the Basic Disk Operating System (BDOS) and the Console Command Processor (CCP). Each of these segments is relatively independent of the others and can be replaced quite easily. However, the prerequisite for this is compliance with defined interface parameters.

The BIOS must exist in some form for every computer. It does all hardware related tasks. All connected devices are controlled by the BIOS as well as the internal hardware. Due to the very different hardware of different computer types, the BIOS is also very different. Usually the BIOS was written by the computer manufacturer. However, there are also some fairly common third-party BIOSes for special computers. ZSDOS has been designed to run on almost any computer whose BIOS is compatible with CP/M 2.2 or ZRDOS 1.x.

The third system segment, the CCP, is the primary interface to the user of the computer. This component is probably the most frequently replaced. The ZCPR family in particular is extremely popular. Since changes in the CCP are most clearly felt by the user, many consider it the most important part of the operating system. In this manual, however, we want to show that the properties of the BDOS determine how flexible and powerful the computer ultimately becomes.

All logical inputs and outputs of the system are controlled by the BDOS. It manages all resources in the form of logical devices, such as Console, printer, and disk drive. The BDOS gives the pure hardware drivers of the BIOS a fixed structure, which creates a uniform appearance of a wide variety of CP/M computers compared to the application programs. This strict separation of hardware-independent routines from hardware-dependent was one of the most significant advances that CP/M brought to the development of operating systems for microcomputers.

ZSDOS is a complete replacement of the BDOS segment and is the subject of this manual.

1.2 Memory allocation

The division of the main memory under ZSDOS is identical to CP/M 2.2 and ZRDOS 1.x systems. The reserved 256 bytes (0 - 0FFH) of the system page are located at the absolute address 0. The TPA (Transient Program Area) is located from address 0100H to the lower end of the lowest system segment. All application programs (e.g. word processing, databases, assemblers, etc.) are executed there. The lower address of the lowest system segment is not determined by ZSDOS.

1.2.1 Storage area of the segments

All program (part) s that are or remain in the system after a warm start are referred to as system segments. Classic representatives for such segments are BIOS and BDOS, which are indispensable for application programs. The CCP can be overwritten by programs and is reloaded each time it returns to the operating system. Some CP/M 2.2 BIOSes reload both the CCP and the BDOS after a warm start. This peculiarity is historical and goes back to a statement by Digital Research that application programs can also overwrite the BDOS if necessary. Practical experience shows, however, that no program (known to us) overwrites the BDOS and most modern BIOSes do not reload the BDOS. For some functions of ZSDOS ("read-only vector received" and "fast re-login") the BDOS must be resident, otherwise they have no effect. (Configuration data will be overwritten)

In addition to the segments just mentioned, there may be other system segments in a ZSDOS system. In contrast to the ones already mentioned, these additional segments are not required to operate a ZSDOS system. However, additional functions can be provided by such segments.

Examples of these segments, which are also known as Resident System Extension (RSX), are: BackGrounder ii and DosDisk, ZCPR3 system components (such as ENV, RCP, IOP, NDR and FCP), utilities such as DateStamperTM and last but not least the ZSDOS extensions for date stamp support for files.

RSXs are usually in memory just below the CCP. All other system extensions are usually located above the BIOS. In summary, the memory allocation is shown in the following overview:

```
              FFFFH +---------------------------+
                    | optional System segments |
              XXXXH +---------------------------+
                    |           BIOS            |
               BIOS +---------------------------+
                    |          ZSDOS            |
         BIOS-0E00H +---------------------------+
                    |           CCP             |
         BIOS-1600H +---------------------------+
                    |      optional RSXs        |
              XXXXH +---------------------------+
                    |                           |
                    |    Transient Program      |
                    |           Area            |
                    |                           |
              0100H +---------------------------+
                    |       System Page         |
              0000H +---------------------------+
```

As can be seen from this illustration, only the addresses of the system page and the start of the TPA area are precisely defined in a ZSDOS system. All other system addresses depend on the BIOS. The sizes of the system page, the ZSDOS and the CCP (except in extended Z systems) are precisely defined. Other system segments can be adapted to your own needs.

1.2.2 System page

The system page (address range from 0 - 0FFH) is used by ZSDOS for important system information. For ZSDOS, the same specifications apply for the system page as under CP/M 2.2. The system page forms the interface to ZSDOS, which is why understanding the function of each individual area is extremely important.

00H - 02H   jump to BIOS warm start routine (BIOS+03H)

No program may change this address. It is the only way to determine the base address of the ZSDOS system segment with certainty. An example of the "correct" use of the warm boot vector follows later in this manual. If programs want to change the BIOS jump vectors, only the values of the BIOS jump table may be adapted, not the jump destination of address 0.

Only Alpha System's NZCOM changes the BIOS warm boot vector in an acceptable manner. A BIOS including jump table is "imitated", which is further down in the memory. The system segments of the

ZCPR are loaded between the real system BIOS and the "imitation" of NZCOM. The BIOS warm boot vector points to the "imitated" BIOS. This means that all programs, except for system-specific utilities, continue to run without errors. When writing or revising a BIOS, one should note the functionality of NZCOM in the system utilities.

03H IOBYTE

The IOBYTE contains a BIOS-dependent structure. It can be used by the BIOS programmer to enable byte-oriented inputs/outputs to be redirected. The byte itself contains 4 fields that stand for the logical device console (CON:), reader (RDR:), punch (PUN:) and list device (LST:). Each logical device can be assigned to one of up to four different physical devices.

Because the integration of the IOBYTE is optional and system dependent, please refer to the manuals of your computer to determine the exact specifications for your system.

04H current CCP default drive and user area

The CCP stores the values for the current default drive and the current user area in the byte of this memory location. The value for the drive is saved in bits 0 to 3, starting with 0 for drive A. Bits 4 to 7 store the user area with modulo 16. Please note that only user areas 0 to 15 can be accessed directly. Since five bits are available for the user area in the directory, files or programs can also be stored in the user areas 16 to 31. From version 3.3 of the ZCPR, logging into user areas 16 to 31 is optionally possible at the command level. The additional CCP code keeps the number of the high user area (with a few exceptional cases).

05H - 07H jump to BDOS

A call to address 5 is used to perform a ZSDOS function. However, the value at address 6 cannot be used as a direct pointer to ZSDOS!

If the size of the available TPA area is required by a program, the jump address stored at addresses 6 and 7 can be used for the calculation. The most significant byte of the value at address 6 always points to the last page of the TPA area. Depending on whether an RSX is loaded or not, the next page contains the ZSDOS segment.

08H - 2FH free for system expansions

30H - 37H reserved

38H - 3FH restart vector 38H

This address is normally used by debuggers to save the breakpoint routine. During the test phase, the debugger only needs one byte (opcode: RST 38H, value: 0FFH) to enter the breakpoint.

40H - 4FH free for system expansions

On some computers, parts of this memory area are used for system functions. For example, on the Ampro Little Board, the ZCPR3 path in stored bytes 40H to 4DH.

50H - 5BH free for programs

Various modem/BBS programs from the public domain area store parameters such as the baud rate.

5CH - 6BH default file control block (FCB) 1

6CH - 7BH default file control block (FCB) 2

The CCP stores the first two parameters of the command line in the file control blocks. When using the first file control block to open a file, the content of the second file control block is overwritten. When opening a file with the second file control block, part of the DMA buffer is overwritten. The 16 bytes of the second file control block should be copied to another memory area before using 6CH and only used there. This means that the file control block at address 5CH can be used unchanged. After opening the file, the complete file control block is 36 bytes (from 5CH to 7FH).

80H - 0FFH default DMA buffer/command line

This memory area is used for two tasks. If a program is started by the CCP, it stores all arguments of the command line for further use by the program. The first byte of the buffer (80H) contains the number of valid characters. The rest of the command line follows itself, starting with the space (on 81H), which separates the arguments from the command.

This area is also used for the default DMA buffer. As long as the address of the DMA buffer has not been changed via the BDOS function 26, all data transfers between memory and disk take place via this memory area. The program must therefore evaluate the command parameters before the buffer is overwritten by disk operations.

2 BDOS functions

The BDOS works not only with floppy drives, but also with other I/O devices, e.g. printer, console, clock, modem etc. Floppy disk I/O is carried out with data blocks, while almost all other devices only transfer single bytes. There are also a number of BDOS calls to read or change data structures and manipulate the file system. As you can see, the BDOS has something more to do than just manage the floppy or file system.

2.1 Character I/O

With character I/O, only one character or byte is transferred at a time. This type of transmission is normally used for the terminal, printer or serial communication (modem) devices. The BDOS incompletely supports four logical devices for character I/O:

* Console (input/output)
* Reader (input)
* Punch (output)
* List output [printer]

In this context, "incomplete" means that the status (ready/not ready) is not available for all logical devices. Because this was overlooked in the development of the original CP/M system segments BDOS and BIOS, the programming of special applications is much more difficult. For example, serial port communication programs to other computer types only work with great effort.

ZSDOS provides nine functions for character I/O. While the first six only provide a direct interface to the corresponding BIOS functions, the last three carry out rudimentary processes. The nine functions are:

* Read a character from the console
* Display a character on the console
* Read a character from the reader
* Output a character to the punch
* Output a character to the list device [printer]
* Query the input status of the console
* Direct console I/O
* Display a character string on the console
* Read console buffer (get character string from console, with editing functions)

2.2 Disk I/O

The floppy disk operations of the BDOS save the data as logical data blocks, which are grouped together to form a file. The file is saved in one of up to 32 user areas of a logical drive. There can be up to 16 logical drives in a CP/M system.

ZSDOS manages the data blocks as 128-byte logical records, as provided by the BIOS. ZSDOS carries out the necessary conversion to determine the physical track and sector numbers. In this way, the physical sector size that is processed in the BIOS remains hidden from the application programs. Regardless of whether the physical sector size is 128, 256, 512, 1024 or 2048 bytes, the BDOS provides a uniform interface. The BDOS offers the following file operations:

* Search for a file
* Rename a file
* Delete a file
* Create a new file
* Open an existing file
* Jump to a specific point in a file
* Read a record from a file
* Write a record to a file
* Close file
* Determine the size of a file

Some of these functions can be applied to closed files. These include: Search, rename, delete, create, open and determine file size. All other functions can only be applied to open files.

2.3 Control and status

The last large category of BDOS functions contains a series of commands for controlling and influencing the status of the system. This includes functions that deliver addresses of data structures. It also defines the functions of connected I/O devices.

Due to the extensions in ZSDOS, many new commands are available in this category (e.g. for error mode, real time clock and date stamp). To make the system even more flexible, some functions can be activated or deactivated by using new control structures in the running system.

Basically, the functions of this category can be divided as follows:

* Return of data structures
  (12 commands)
* Define control elements
  (7 commands)
* System control
  (6 commands)
* Interface to the real time clock
  (2 commands)
* Time and date stamp for files
  (2 commands)

3 ZSDOS Data Structures

3.1 General

Various data structures are used to exchange data between ZSDOS and application programs. When the program transfers information to ZSDOS, a ZSDOS function is called and a byte value in register E or a word pointer to a data structure in register pair DE is transferred.

Passing multiple values to ZSDOS is only a little more difficult. Normally, no special precautions are necessary. If pointers to certain data structures are passed in repeated calls, the data structure should not be shifted between calls because the address of the structure is passed to ZSDOS each time. An example: If a file with the file control block (FCB) is opened at a certain address, the FCB should not be moved to another address. Failure to do this can lead to problems with DateStamperTM, BGii or other programs.

If information from ZSDOS is returned to the program, it is available in registers (byte or word), in the previously defined buffer area, in the current DMA buffer or as a pointer to the current data area. As far as possible, the same structures are used for the transfer of information to and from ZSDOS. For example, the file control block (FCB) corresponds largely to the data structure with which the directory information is stored on disk. If entries are to be made in the directory, the fields of the FCB are initialized by the application program that correspond to the directory fields.

3.2 Logical record

The most basic structure used by ZSDOS is the logical record. It contains 128 bytes of data/information read from or written to a floppy disk. Please note that the term "logical record" should not be confused with the term "sector". Normally, a sector of the disk contains several logical records.

With the disk functions "Read" or "Find first/next file", ZSDOS reads the information of a logical record in the current DMA buffer. In the same way, write to disk from there using the write functions. The base address of the DMA buffer is defined via the function call 26, the desired address being transferred in the register pair DE. Function 47 can be used to query the current DMA address, which is returned in register pair HL.

3.3 File Control Block (FCB)

Another basic structure is the file control block (hereinafter referred to as FCB), which is used to transfer information to ZSDOS for most file-related functions. The FCB is a 36-byte data area that contains information required for file manipulation. The FCB is structured as follows:

FCB+00H	Drive number (DR, 0=default drive, 1...16=A-P)
FCB+01H	File name in uppercase ASCII letters [8 bytes] (Fn)
FCB+09H	ASCII uppercase file type [3 bytes] (Tn)
FCB+0CH	Logical extent number (EX)
FCB+0DH	User area (S1)
FCB+0EH	Data module (S2)
FCB+0FH	Extent record counter (RC)
FCB+10H	16 byte disk map for this extent (AL)
FCB+20H	Current record for R/W (CR)
FCB+21H	Random access record number LSB (Rn)
FCB+22H	Random access record number ISB
FCB+23H	Random access record number MSB

The most significant bits in the file name and in the file type are used to store the attributes. Such attributes mark a file as read-only or archived. For more information on using file attributes, see function 30 in section 5.2.30.

The logical extent (EX) is used by ZSDOS in order to be able to process files that are larger than 128 logical records (16 kB). For the first logical extent, this number is set to 0 and increased by one each time for a further 128 records.

The data module number (S2) is used by ZSDOS to process files that are larger than 32 logical extents (512 kB). It is also initially set to 0 and is increased by one each time for a further 64 logical extents.

These fields of the FCB are normally set to zero by the application programs, but in certain cases are set to different values. The logical extent number can take values between 0 and 31. Values between 0 and 63 are permitted for the data module number. You can find more detailed information in section 5.2.17 (Description of functions "Search for First File") of this manual.

The last record used number is stored in the current logical extent in the byte of the record counter. The allocation vector represents the number of blocks occupied by the file. No data from any of these fields is passed to ZSDOS.

The next record number is stored in the current extent in the current record byte, which is accessed by the functions "Sequential Read" or "Sequential Write". Normally, this byte is set to zero by the user when searching or opening files. It should not be changed between sequential write or read accesses.

The three-byte field with the number for the random record is used by the functions "Random Access Read" and "Random Access Write". The 24 bits can contain a number between 0 and 262,143. This results in the maximum size of a file of 262,144 records.

There is a very important difference between ZSDOS and other DOS segments for CP/M 2.2. ZSDOS only opens or creates new extents when they are needed and not when the last record of an extent is read or written. This means that after the sequential reading or writing of the last record of an extent, the bytes of the current record and the record counter are set to 80H. Some older application programs are unable to cope with this functionality of the DOS and therefore do not run under ZSDOS. Because CP/M Plus deals with extents in exactly the same way as ZSDOS, these programs do not run under CP/M Plus.

It is extremely important to be aware of the importance of the FCB for DOS in file operations. All status information is saved there. If a program tries to change the FCB of an open file, the entire DOS status will be destroyed! This is especially true when DosDisk is running and the FCB is in MS-DOS format. It is therefore strongly recommended that applications should never change the FCB fields of an open file. The only exceptions are the fields of the current record and the number for the random record.

3.4 Directory Record

The directory record is a data structure that the BDOS writes to disk. This contains information about the current assignment of the files. Each directory record is 128 bytes long (a logical record) and usually contains four directory entries. There is at least one directory entry for each file on the disk. If a file is so large that several directory entries are required for it, such an entry is referred to as a "physical extent". Each physical extent has its own number, so that ZSDOS can correctly access the files.

Each directory entry is 32 bytes long and is structured similar to the file control block. The main difference is that the fields for the current record and for the random record are not available in the directory entry. In addition, the user area of the file is saved in the first byte of the directory entry rather than the drive code or the value 0E5H if the file was deleted.

Directory entries are structured as follows:

DIR+0	User area of the file (0..31)
DIR+1..8	File name in uppercase ASCII letters [8 bytes]
DIR+9..11	ASCII uppercase file type [3 bytes]
DIR+12	(EX) Logical extent number
DIR+13	(S1) system byte (set to 0)
DIR+14	(S2) data module
DIR+15	(RC) Record counter
DIR+16..31	(AL) Allocation vector for this physical extent

As with the FCB, the most significant bits of the file name or type are used to store the attributes.

3.5 Disk Allocation Vector

The disk allocation vector is a structure of the BDOS that is embedded in the BIOS. There is an allocation vector for each logical drive in the system. It is a bit map of all blocks on the floppy disk. If a bit is set to 1, this means that the assigned block is used (occupied). Accordingly, a bit reset to 0 indicates that the block is available. The minimum size of the allocation vector in bytes for a drive can be calculated as follows: number of blocks/8+1.

Access to this structure is very unusual for an application program (apart from some directory programs that derive the available disk space). Application programs must never change the allocation vector directly. Function 27 returns the address of the allocation vector for the current default drive in register pair HL. However, we recommend that the vector should not be accessed, as this may not be possible in future systems. This is, for example, already the case with CP/M Plus and may also be the case in future versions of ZSDOS.

3.6 Disk Parameter Block

The disk parameter block (DPB) is a BIOS structure that defines the format of a logical drive for ZSDOS. If a program needs information from the DPB, it must access it directly. The DPB is structured as follows:

DPB+0,1	Number of 128 byte records per track
DPB+2	Block shift factor
DPB+3	Block mask
DPB+4	Extent mask
DPB+5,6	Number of the maximum available blocks
DPB+7,8	Number of directory entries -1
DPB+9,10	Bit map of the directory and reserved blocks
DPB+11,12	Size of the directory check buffer
DPB+13,14	number of system tracks

A detailed description of the individual fields of the DPB would go beyond the scope of this manual. We therefore recommend the very good book "The Programmer's CP/M Handbook" by Andy Johnson-Laird (see the bibliography in the ZSDOS User's Guide). Function 31 returns the address of the DPB for the default drive in register pair HL.

3.7 Dates

If the driver routines to support date stamps are installed, ZSDOS supports two other structures. The first is the date specification, which is used to exchange time and date information between application programs and ZSDOS. The date is based on a series of packed BCD numbers and is structured as follows:

TIME+0	last 2 digits of the year (from 78 to 99 is preceded by 19 for the century, otherwise 20)	
TIME+1	month	[1..12]
TIME+2	Day	[1..31]
TIME+3	hour	[0..23]
TIME+4	minute	[0..59]
TIME+5	Second	[0..59]

Anyone familiar with DateStamperTM will immediately recognize this format. The only deviation from the DateStamperTM format is the preceding century. DateStamperTM always assumes "19" for the century. For more information about the DateStamperTM format, you should refer to the DateStamperTM manual starting on page A-19.

When function 98 is called, ZSDOS returns the current time in the buffer, the start address of which is passed in the register pair DE. Calling function 99 sets the clock to the values in the buffer, the start address of which is transferred in register pair DE.

The relative clock known from DateStamperTM is also supported. Only the hour and minute fields are required for this, the seconds field is set to zero. The relative clock is just a simple binary counter. The minute field is used as the low-order byte, while the hour field is the high-order byte. To avoid confusing the relative clock with a real-time clock, the most significant bit (bit 7) is set in the hour field for a relative clock.

3.8 stamp format

The stamp format for files used by ZSDOS was also derived from DateStamperTM. (Honestly folks, we didn't want to plagiarize Bridger Mitchell's ideas. We carefully examined each existing format before deciding on the method that best suited our purposes.) Regardless of the stamp method used in the system - DateStamperTM or P2DOS or CP/M Plus format - the format used internally is always the same.

The universal format consists of three fields that contain information about the creation, the last access and the last change. The first 5 bytes of the date are used for this. If a field is not supported or is only partially supported, the corresponding areas in the field are set to 0 when reading and ignored when writing.

All stamp information is transferred in the current DMA buffer.

Function 102 returns the date stamp information of a file, the FCB address of which was transferred in the register pair DE. Function 103 is used to transfer the stamp information from the DMA buffer to a file whose FCB address in the DE register pair. At the moment, the stamp information is only 15 bytes long. However, this could change in future versions of ZSDOS, so that for compatibility reasons we recommend reserving a 128-byte buffer in the application program.

Stamp format for files (15 bytes packed BCD numbers):

DMA+0..4	Created date	(first 5 bytes of the date)
DMA+5..9	Last Access date	(first 5 bytes of the date)
DMA+10..14	Modified date	(first 5 bytes of the date)

4 ZSDOS programming conventions

4.1 General

ZSDOS is compatible with CP/M 2.2 and ZRDOS applications. However, to create a solid foundation for future extensions of ZSDOS and compatible BDOS substitutes, the following practices should be considered during programming.

Under ZSDOS it is assumed that the system page has the same structure as under CP/M 2.2 or ZRDOS - i.e. a jump to the warm start routine of the BIOS (jump target: BIOS+3) at address 0000H and a jump to the BDOS or the lowest resident system extension (RSX) at address 0005H. A correct system address cannot be derived from the jump destination of address 0005H, except for the end of the TPA area. The following lines of source text are intended to illustrate how the upper end of the TPA area can be calculated using the jump instruction at address 0005H:

```
GETSIZ: LD      HL,(0006H)      ; Get the end address of the TPA
        DEC     HL              ; correct address now in HL
        ...
```

This address is often used by application programs to calculate the start addresses of BDOS and BIOS. However, this method does not work if resident system extensions (RSXs) such as BGii, ZEX, DosDisk etc. are installed.

To correctly calculate the system addresses, the jump destination of address 0000H must be used. The pointer of this address always points to BIOS+3 and should never be changed by any program. When programs need to intercept the BIOS entry points, e.g. for warm start, console status etc., the jump table of the BIOS should be changed and not the jump destination at address 0000H. The ZSDOS entry point can be calculated correctly according to the following example:

```
FINDZS: LD      HL,(0001H)      ; Get BIOS warm start address
        LD      DE,-0DFDH       ; Offset to start of ZSDOS
        ADD     HL,DE           ; HL points to ZSDOS entry point
```

As with CP/M 2.2, ZRDOS up to version 1.9 and most unbanked systems, the base address of ZSDOS is 0DFDH below the jump destination at address 0000H. The start address of the CCP can also be calculated in this way. To do this, just subtract the value 1603H from the jump target. The start of the BIOS jump table (cold start) is obtained by subtracting 3 from the jump target.

If you now combine the "correct" calculation method of the BDOS entry point and the jump destination at address 0005H, you can easily determine whether an RSX is installed.

```
FINDZS: LD      HL,(0001H)      ; Get BIOS warm start address
        LD      DE,-0DFDH       ; The beginning of ZSDOS below
        ADD     HL,DE           ; HL points to ZSDOS entry point
        EX      DE,HL
        LD      HL,(0006H)      ; Get BDOS jump vector
        AND     A
        SBC     HL,DE           ; are the addresses identical?
        JR      Z,NORSX         ; yes - no RSX available
        ...
```

In ZCPR systems with an extended environment, the BIOS, BDOS and CCP addresses are available in the environment. If an extended environment is available, all system addresses must be taken from the environment. The reason for this may be "abnormal" sizes of BDOS and CCP of such systems.

ZSDOS functions are called up by storing the ZSDOS function number in register C and values in register E or register pair DE. A call (CALL) is then made to address 0005H. ZSDOS returns values in register A and/or in register pair HL. All register contents with the exception of IX, IY and the alternative registers can be changed.

When developing BIOSes, BIOS extensions or IOPs with reentrancy, it should be borne in mind that all registers that are not included in the original 8080 register set must be backed up and restored between calls. The application programs "owns" the registers AF', BC', DE', HL', IX and IY - not the system software! According to the conventions, however, registers I and R are subject to the BIOS. With new processors like 64180 and Z280, all new registers (with the exception of the Z280 SSP) belong to the BIOS because they are hardware-specific and directly related to inputs and outputs. The Z280 SSP should remain reserved for the BDOS.

Many programmers have used techniques in file operations that cause problems with advanced operating systems. For example, the search for files is affected, which under certain circumstances can lead to apparent malfunctions with ZSDOS when the path is activated. The test for the existence of a file should be carried out using the BDOS function 17 provided for this purpose and should not be based on the evaluation of the returned code of the opening function. When opening, ZSDOS searches for the specified file along the path, but not when searching for the first entry (function 17).

In addition, programmers have to take into account that with ZSDOS much larger files can be created than under CP/M 2.2 or ZRDOS. While the latter systems only support file sizes of up to 65,536 logical records (8,192 kB), files under ZSDOS can be up to 262,144 logical records (32,768 kB). Such files can also be processed by CP/M 3.0.

4.2 Reentrancy in ZSDOS calls

Reentrant ZSDOS function calls in the sense of the ZRDOS 1.x specifications are fully supported by ZSDOS. This property is generally only used by ZCPR I/O packages (IOPs).

A reenterant ZSDOS function call is triggered when ZSDOS calls a BIOS function that is intercepted by an IOP, which in turn calls a ZSDOS function. An example of this would be a printer spooler (a program that redirects printer output to a file). ZSDOS would call the BIOS function for the list output. The IOP intercepts this function call and then calls ZSDOS functions to write the intercepted data to a file.

The possibility of reentrancy is a very powerful tool, but it can also cause unprecedented damage in the system! All important internal status information from ZSDOS (including the address of the DMA buffer) must be saved before each function call and then restored again. In addition, some BIOS information must also be saved or reinitialized. A basic rule is: No BDOS disk function may be interrupted.

In order to offer maximum compatibility with existing programs for ZRDOS Plus, the data areas of ZSDOS were located in the lower part of the system segment. The addresses used and the method of reentrancy are compatible with existing IOPs for ZRDOS Plus. Here is a comparison of the ZSDOS parameters with the requirements of ZRDOS Plus:

	Beginning	length
ZSDOS	Base+3	146 (92H) bytes
ZRDOS Plus	Base+5	147 (93H) bytes

Because the data area of ZSDOS is smaller, there is no damage if the 147 bytes required by ZRDOS Plus are saved. The different start addresses are due to the fact that in ZSDOS there is an internal error vector table on a CP/M 2.2 compatible offset. The address of the BAD SECTOR error routine is located on the affected bytes. The user should make sure that no routine at ZSDOS reentrancy with ZRDOS parameters changes the BAD SECTOR error vector. Otherwise strange things could happen.

As already shown, the base address of ZSDOS can be easily calculated using the BIOS warm start vector at addresses 0001 and 0002. Only 0DFDH has to be subtracted from the vector to get the "base".

One last precautionary measure must be taken when using the reentrancy options in the event that traps in the BIOS jump vectors are used to initiate reentering ZSDOS calls: The user must ensure that all registers, including the IX register, are saved between the reentering calls!

Example:

To integrate a reentrant function call, the address of the DMA buffer must first be queried by the DOS and saved in your program. Then the data area of the DOS must also be saved in your program. From this point on, all function calls can be made regardless of the previous state of the DOS. Once your routine has been executed, you should restore the DOS by reversing the steps at the beginning; first the saved data area is copied to its original position and then the address of the DMA buffer is set to the old value using function 26. The address must be restored with the DOS function 26 in order to compare the DMA address of the BIOS with that in the ZSDOS data area, if it is changed by your program.

```
        LD      C,47            ; Get current DMA address
        CALL    5               ; ... call the DOS entry point
        LD      (DMASAV),HL     ; Save DMA address locally
        CALL    FINDZS          ; Find ZSDOS base address
                                ; ... (see section 4.1)
        LD      DE,9-6          ; Offset of the data area
        ADD     HL,DE           ; ... from the DOS start
        LD      DE,SAVAREA      ; Local backup area pointer
        LD      BC,147          ; ... and the whole area
        LDIR                    ; ... copy the block
        ...                     ; here is your routine
        CALL    FINDZS          ; Find ZSDOS base addr again
        LD      DE,9-6          ; Offset of the data area
        ADD     HL,DE           ; ... from the start of DOS
        EX      DE,HL           ; in the reg. for the goal
        LD      HL,SAVAREA      ; Pointer to source on the
                                ; ... the local security area
        LD      BC,147          ; the whole area
        LDIR                    ; ... copy the block
        LD      DE,(DMASAV)     ; Get secured DMA address
        LD      C,26            ; ... and with function 26
        CALL    5               ; ... set via DOS (and in BIOS)
        ...                     ; go on...
```

4.3 ZSDOS configuration area

The configuration area of ZSDOS is located at address ZSDOS Base+3. Various vector tables, the addresses of the path and the wheel byte as well as the configuration byte of the flags are stored in this area. The same assignment is retained for all versions 1.x of ZSDOS and ZDDOS. However, changed offsets may be used in future publications.

4.3.1 Error vector table

The error vector table is located at address base+3. This is a CP/M 2.2 compatible structure that is used by some programs (e.g. sector check programs) to intercept the BDOS errors. It was only integrated into ZSDOS because of the compatibility with existing programs. ZSDOS applications should use the new BDOS error modes to intercept errors.

4.3.2 Path address (ZSDOS only)

The base address of the search path is stored at address base+11. When opening files, ZSDOS uses the search path if the value at this address is not equal to zero and bit 5 of the configuration byte is set to 1.

Please note that if the CCP uses a command search path (as with ZCPR or BGii), the DOS path is searched as often as there are entries in the command search path of the CCP. Of course, this slows down the work. Future versions of these replacement CCP systems should check for a ZSDOS path to initiate one of the following procedures. If it is determined that a DOS path is available and activated, this is used instead of the CCP search path. The second option would be to deactivate the DOS path via bit 5 of the flag byte during the search along the CCP command search path.

4.3.3 Address of the wheel byte

The address of the wheel byte is saved in base+13. The wheel byte is a ZCPR control element with which the safety functions of the system can be expanded. If the wheel byte is off (value equals 0), ZSDOS protects all files with the wheel protection attribute (f8) set from being overwritten, deleted and renamed. If the address of the wheel byte in the BDOS is set to 0 (pointer to the jump to warm start), ZSDOS assumes that the user has all rights of use and allows him unrestricted access to the files.

Please note that the wheel byte is an element of ZCPR3 and any arbitrary address can be set for the independent DOS control mechanism. If you use one of the ZCPR versions 3.x as a replacement for the CCP, you will certainly use the same wheel byte for both system segments. We just want to point out that different memory cells are possible for special installations.

4.3.4 Configuration byte

Many properties of ZSDOS can be checked during runtime by changing the configuration byte. The byte is based on address+15. Not all flag bits are used by ZSDOS. If unused bits are set or reset, this has no effect on ZSDOS.

To ensure compatibility with later versions of ZSDOS, only the functions 100 (hole flags) and 101 (set flags) should be used to access the ZSDOS flags. The meaning of the flag bits in the configuration byte is defined as follows:

```
Bit:   7 6 5 4 3 2 1 0
       | | | | | | | +- Public files               on (1)/off (0)
       | | | | | | +--- Write public/path files    on (1)/off (0)
       | | | | | +----- Get read-only vector       on (1)/off (0)
       | | | | +------- quick login                on (1)/off (0)
       | | | +--------- Floppy disk change warning on (1)/off (0)
       | | +----------- ZCPR2/3 path               on (1)/off (0)
       | +------------- Path with/out system files on (1)/off (0)
       +--------------- reserved
```

Bit 0 controls whether ZSDOS finds files with the "public file" (f2) attribute in other user areas on the same disk. If the bit is set to 1, such files are found if a unique file name has been specified.

Bit 1 decides whether it is allowed to write to files that were found using the public attribute or the path (only ZSDOS). If the function is switched on (bit 1 equals 1), files can be written to. Otherwise (bit 1 equals 0) writing to files that were found using the attribute public or the path is not possible.

Bit 2 specifies when the write protection vector in ZSDOS is deleted. If the bit is set, the vector is never deleted (as long as ZSDOS is not reloaded from the disk). If the bit is reset, the write protection bit for a drive is deleted when it is logged in again with function 13 or 37.

Bit 3 causes ZSDOS not to recreate the allocation vector of a "fixed" disk (hard disk or RAM disk) if the drive has been logged out with function 13. Setting this bit speeds up work with hard disks considerably. Fixed disks can be logged in again at any time using function 37.

Bit 4 switches the message from ZSDOS on or off when changing floppy disks. If the bit is set, ZSDOS issues a message on the screen each time a disk is changed. This message is of course suppressed if the BDOS error mode is set accordingly; regardless of the state of this bit.

Bit 5 enables the use of the DOS path when opening files if it is set to 1 (ZSDOS only). If this bit is set and a non-zero value is entered in the address for the path, the specified file with a unique name is found by ZSDOS using the path. This bit has no meaning for ZDDOS.

Bit 6 specifies the path access if the path is activated (ZSDOS only). If bit 6 is set to 1, all files in directories along the path are found regardless of the system attribute (path directory access). If this function is switched off (bit 6 equals 0), only files with the system attribute set (path file access) are found in the directories along the path. This bit has no meaning for ZDDOS.

4.3.5 Date vectors

A vector table is integrated in ZSDOS, which allows drivers for date stamps to integrate themselves into ZSDOS. The table contains 6 entries and starts at address base+16. Another dummy entry is only used to record the address of the switch-off function in ZSDOS.

The special drivers for date stamps install themselves in ZSDOS by entering the addresses of the supported functions in the table. ZSDOS calls the required routines to perform the date stamp functions. Before doing so, however, the directory buffer is updated and a check is carried out to determine whether the disk has read/write status.

Because ZDDOS already contains DateStamperTM, only the addresses for the clock driver, for the removal and for the dummy entry are necessary for this DOS.

The structure of the date vector table:

Base+16	Vector of the routine for reading/setting the RTC
Base+18	Vector of last access stamp routine
Base+20	Vector of the routine for creation stamp
Base+22	Vector of the routine for the modified stamp
Base+24	Vector of routine to get stamp
Base+26	Vector of routine to put stamp
Base+28	Vector of the dummy routine
Base+30	Address of the routine for removing the date stamp

4.4 Routines to support time and date stamps

ZSDOS and ZDDOS differ significantly with regard to the routines for supporting time and date stamps. DateStamperTM is already integrated in ZDDOS, so that only one clock driver is required. ZSDOS does not include a stamp routine. Both an external clock driver and an external stamp routine are required for operation.

Different forms of date stamps are possible with ZSDOS. Supported methods include Plu*Perfect's DateStamperTM and P2DOS (compatible with CP/M Plus) date stamps. As long as no corresponding routine for supporting date stamps is installed in the ZSDOS system, the date stamps cannot be activated. It is not necessary to install such routines to operate ZSDOS. They are only required if you want to use functions 98, 99, 102 and 103.

Depending on the desired stamping method, different routines are required. With the integrated DateStamperTM support, ZDDOS only allows the use of this method. With ZSDOS, DateStamperTM, P2DOS or both types of stamps can be used. Drivers for other stamping methods can be programmed and easily integrated into ZSDOS. For this purpose, ZSDOS only provides defined connection points, while the actual routine is located outside the BDOS segment, typically above the BIOS. It is based on the same philosophy as ZCPR - optional parts of the system are moved to reserved buffer areas above the BIOS.

ZSDOS does most of the detail work for the routines. The requested FCB is copied into the directory buffer, the disk is checked for read/write status if necessary, the DMA buffer and the offset of the directory buffer are provided according to the method.

Due to the close connection of the DOS with the routines for date stamping, the average memory requirement for DateStamperTM with a real-time clock driver under ZSDOS is approx. 3/4 kByte - much less than with earlier DateStampers. With ZDDOS and the integrated DateStamperTM, the memory requirement is reduced even further, since only the clock driver is missing. This is usually less than 400 bytes. If a clock driver is already available in the system, no additional storage space is required under ZDDOS.

ZSDOS comes with special drivers for DateStamperTM, P2DOS (CP/M Plus compatible) and to support both formats. The drivers for both methods each read a format and write both. They are particularly interesting for users who require the highest level of compatibility between CP/M Plus and ZSDOS systems.

5 ZSDOS function calls

5.1 Description of the returned values

The ZSDOS functions return values to indicate the success or errors that occurred when executing the function. There are five categories of these codes - directory codes, error codes, time/date codes, read/write codes and extended error codes. The following overview shows the returned values of the codes of each category:

Directory code:

```
        A = 00H, 01H, 02H, 03H if no error has occurred
        A = 0FFH, in the event of an error
```

Error code:

```
        A = 00H, no error
        A = 0FFH, error occurred
```

Time/date code:

```
        A = 01H if no error has occurred
        A = 0FFH, error occurred
```

Read/write code:

```
        A = 00H, if no error has occurred
        A = 01H, read - end of file write - directory full
        A = 02H, floppy disk full
        A = 03H, error while closing on random read/write
        A = 04H, empty record for random reading
        A = 05H, directory full of random writing
        A = 06H, random access record number during random
                 reading/writing too large
```

extended error codes in error mode:

```
        A = 0FFH, further error codes in H
        H = 01H, disk I/O error (defective sector)
        H = 02H, floppy disk write-protected (read only)
        H = 03H, file read-only
        H = 04H, illegal drive selected
```

The following function descriptions show which values are returned by each function. The only exception is the extended error codes that are returned by any function that performs disk access. However, these extended codes are only returned if one of the two modes for returning the error code has been set using function 41.

5.2 Functional description

```
+----------------------------------------------------------------+
|                Function 0 - Terminate Program                  |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   none                        |   none                         |
+-------------------------------+--------------------------------+
```

This call, which is rarely used, clearly distinguishes it from application programs. If this function is called, ZSDOS executes a RST 0 command internally. In a ROM-based ZSDOS system, the RAM data segment is initialized and loaded with the default values.

Most programmers use an RET command (if the CCP has not been overwritten) or a jump to address 0 to end their program. Function 0 is a one-way street - it does not return to the calling program.

The result of this call corresponds to the warm start of the system - all drives with exchangeable media are reset, the DMA address is set to 80H, the CCP is reloaded (unless it is protected by an RSX) and control is transferred to it. In addition, the ZSDOS error mode is reset to the default by calling this function.

```
DONE:   LD      C,0
        CALL    BDOS            ; One-way street - no way back
```

Function 1 - Console Input Byte	
Input: no	Output: A = character

This function returns the next character from the console. If no character is available when this function is called, ZSDOS waits for an input before returning to the calling program. The returned character is output on the console, with control characters being filtered.

Carriage return (0DH), line feed (0AH) and backspace (08H) are reproduced unchanged. All tab stops (09H) are converted to the corresponding number of spaces in order to position the cursor on the next column that can be divided by 8. All other control characters are not displayed on the console.

Control-S is intercepted by this function and treated as follows: If a Control-S was detected, all outputs to the console are stopped until any other character (except Control-C) is entered. After that, console output continues. If a Control-C is recognized after entering Control-S, ZSDOS resets the error mode to the default mode and then carries out a warm start.

```
CONIN:  LD      C,1
        CALL    BDOS            ; next char from the console
        ...                     ; Character is now in register A
```

Function 2 - Console Output Byte

Input:	Output:
E = character	none (A = BIOS A register)

The character contained in register E is output to the current console device with this function. As with function 1, all tab stops are converted to spaces, so that the cursor is positioned on the next column that can be divided by 8.

The console input function controls this function when a Control-S occurs. If a Control-S has been entered, the output to the console will be blocked until any other character (except Control-C) is entered. If a Control-C is entered after entering Control-S, ZSDOS resets the error mode to the default mode and then carries out a warm start.

Note: This function should not be used to output video control characters because certain control characters are filtered. Function 6 should be used for these purposes.

```
CONOUT: LD      E,A             ; suppose the char is in A
        LD      C,2             ; Select function console output
        CALL    BDOS            ; Send characters to the console
        ...
```

```
+----------------------------------------------------------------+
|                    Function 3 - Reader Input                   |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   none                         |   A = character               |
+--------------------------------+-------------------------------+
```

This function fetches the next character from the current reader input device (RDR:). If no character is available when this function is called, ZSDOS waits for an input before returning to the calling program. If no device is defined for reader input, the returned value depends on the dummy routine of the BIOS. Control characters are not filtered by this function call.

In earlier BDOS systems such as CP/M 2.2, the "paper tape reader" was defined as the device for the reader input. This term comes from the time when paper tape was a common medium for data storage.

The input from the reader device could be something like this:

```
AUXIN:  LD      C,3
        CALL    BDOS            ; next char from the reader
        ...                     ; Character is now in register A
```

Function 4 - Punch Output	
Input: E = character	Output: none (A = BIOS A register)

The punch output function sends the character in register E to the current punch output device (PUN:). Before the character is sent, the function waits for the device to be ready.

In earlier BDOS systems such as CP/M 2.2, paper tape was a common media for data storage, and the "punch" was a paper tape punch.

A character can be sent to the punch output device as follows:

```
AUXOUT: LD      E,A             ; suppose the char is in A
        LD      C,4             ; Select punch output
        CALL    BDOS            ; ... and send characters
        ...
```

```
+----------------------------------------------------------------+
|                  Function 5 - List Output Byte                 |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   E = character                |   none (A = BIOS A register)  |
+--------------------------------+-------------------------------+
```

The character contained in register E is transferred to the current list device (LST:). The BIOS function for list output waits for the device to be ready before the character is transferred and returned to the BDOS. The BDOS does not call the BIOS routine for querying the list output status before sending the byte.

```
LIST:   LD      E,A             ; suppose the char is in A
        LD      C,5             ; Select list output
        CALL    BDOS            ; ... and send characters
        ...
```

Function 6 - Direct Console I/O

Input:	Output:
E = 0FFH (input)	A = input char (00=none)
E = 0FEH (input)	A = cons. status (00 = none)
E = 0FDH (input)	A = input char
E = 0..0FCH (output)	none (A = BIOS A register)

This function call (sometimes also called DCIO for short) is used to bypass the normal filtered input and output of the BDOS and to communicate directly with the console via the BIOS routines. Normally, video control sequences are transmitted to the terminal with this function.

In ZSDOS some shortcomings of the CP/M 2.2 BDOS were fixed, where calls of function 6 were mixed with the normal BDOS console input of function 1. This affects the internal character buffer of the DOS if Control-S characters are used to start and stop the screen output. Each time function 6 is called for character input or status query, ZSDOS checks the character buffer in order to always provide correct results when reading the console. At this point we would like to thank Bridger Mitchell, who pointed out the peculiarity of CP/M so that we were able to eliminate it.

With the value in register E, function call 6 in CP/M 2.2 and ZSDOS systems determine the console function to be performed. All returned values are made available in register A. The following values are defined for register E:

0FFH   get the next character from the console or 0 if no character is available

0FEH   query the status of the console; 0 indicates that no character is available

0FDH*   wait for the next character from the console and return it

0..0FCH* Output of the character in register E on the console

* = new or changed functions in ZSDOS

The function added (0FDH) corresponds to that of CP/M Plus and provides a more convenient routine for "get next character".

```
OLDCODE:LD      E,0FEH          ; Query console status
        LD      C,6
        CALL    BDOS
        AND     A               ; anything arrived?
        JR      Z,OLDCODE       ; ... no, repeat
        LD      E,0FFH
        LD      C,6
        CALL    BDOS            ; finally ready, pick it up
        ...                     ; Characters now in A

; new method with ZSDOS ...

NEWCODE:LD      E,0FDH          ; get next character
        LD      C,6             ; as soon as it's there
        CALL    BDOS
        ...                     ; Character now in A
```

Function 7 - Get IOBYTE

Input:	Output:
none	A=IOBYTE (system page+03H)

This function returns the value of the current IOBYTE in register A. The IOBYTE is an optional and BIOS-dependent structure. It can be used to redirect byte-oriented I/O of the logical devices CON:, RDR:, PUN: and LST:. Please refer to your computer's manuals to determine the exact specifications for your system.

Note: This function call may no longer be available in future ZSDOS versions.

```
GETIOB: LD      C,7             ; get IOBYTE
        CALL    BDOS            ; ... is returned in A.
        ...
```

```
+----------------------------------------------------------------+
|                    Function 8 - Set IOBYTE                     |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   E = IOBYTE                  |   none (A = IOBYTE)            |
+-------------------------------+--------------------------------+
```

This function sets the IOBYTE to the value in register E. The IOBYTE is an optional and BIOS-dependent structure. It can be used to redirect byte-oriented I/O of the logical devices CON:, RDR:, PUN: and LST:. Please refer to your computer's manuals to determine the exact specifications for your system.

Note: This function call may no longer be available in future ZSDOS versions.

```
SETIOB: LD      E,A             ; suppose IOBYTE is in A
        LD      C,8
        CALL    BDOS            ; set IOBYTE
        ...
```

```
+-----------------------------------------------------------------+
|              Function 9 - Console Output String                 |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   DE = address of the string,  |   none (A = '$')               |
|   that ends with '$'           |                                |
+--------------------------------+--------------------------------+
```

A character string consisting of ASCII characters that ends with a dollar sign '$' is output with this function on the console. All characters in the chain with the exception of the dollar sign are transferred to the console. All tab stops are converted to a corresponding number of spaces in order to position the cursor on the next column that can be divided by 8.

The console is checked for the input of Control-S while the string is being output. In this case, the output is stopped until another character is entered and then continued. If Control-S is followed by Control-C, ZSDOS resets the error mode to the default and then performs a warm start. In this way, the user can end faulty programs without performing a cold start.

```
STROUT: LD      DE,STRING       ; what you want to display
        LD      C,9             ; Select string output
        CALL    BDOS
        ...
STRING: DEFB    'This is a string. $'
        ...
```

```
+-----------------------------------------------------------------+
|                Function 10 - Console Input Line                 |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   DE = address input buffer    |   none (A = 0DH)               |
+--------------------------------+--------------------------------+
```

This function returns a character string from the current input device of the console. The caller myst pass the pointer to an input buffer. This buffer is configured as follows:

BUFF+0 size of the buffer for the maximum number of characters to be read (maximum 255)
BUFF+1 actual number of read characters (set by BDOS when returning)
BUFF+2 up to max. Length+2 characters from the console

Some control characters are available in function 10 for editing the input line:

^H   deletes characters to the left of the cursor

^J   ends the entry

^M   ends the entry

^X   deletes the entire line

^U   as ^X

^R   rewrites current line (ZSDOS only)

DEL   as ^H

All tab stops are converted to spaces by function 10 (as with functions 1, 2 and 9), but only for screen output - ^I is retained in the buffer. ZSDOS remembers the cursor position when this function was called, so that tabs are expanded properly (as long as something was not output with function 6 on the same line). The non-printable control characters (all except tab and edit control characters) are converted into two characters for the screen output. The first character is a caret '^', followed by the control character+40H. For example, Control-Z would be displayed as '^Z'.

The function recognizes Control-P and switches the printer flag accordingly. Control-S (to stop console output) is not recognized by this function.

Function 10 ends in the following cases:

1. Enter (Carriage Return) or line feed was pressed.

2. The input buffer is full.

3. If the first character entered in the line is a Control-C, the program is terminated and the system is restarted. In this case the Control-C remains in the buffer so that the ZCPR command processor does not get mixed up.

When returning to the calling program, the number of characters read is entered in BUFF+1. The character string itself starts at BUFF+2. Note that Enter or linefeed are read, but do not appear in the buffer.

An example of using function 10:

```
BUFFRD: LD      DE,BUFF         ; Pointer to the text buffer
        LD      C,10            ; Read console buffer function
        CALL    BDOS
        ...
                                ; Structure total of 128 chars
BUFF:   DEFB    126             ; Max.B 126 characters are read
        DEFB    0               ; actual number here from ZSDOS
        DEFS    126             ; actual buffer area
        ...
```

Function 11 - Console Status Check

Input:	Output:

none	A = 0, no character
	A = 1, character available

This function is used to query the console device whether a character has been entered. ZSDOS returns the value 0 in register A if no character is available or the value 1 in the other case.

```
CONST:  LD      C,11            ; Check console status
        CALL    BDOS            ; returns status in A.
        AND     A               ; something available?
        JR      Z,NOCHAR        ; ... jump if there is no char.
        ...
```

Function 12 - Get System Identification	
Input: none	Output: HL = 22H (CP/M compatible)

This function returns the value 22H in the HL register pair to indicate compatibility with CP/M 2.2. Function 48 must be used to query the version of ZSDOS.

In the case of ZDDOS or ZSDOS with DateStamperTM installed, if register D contains the value 'D' (044H) when this function is called, the address of the DateStamperTM is returned in register pair DE and the ASCII character 'D' is returned in register H. This functionality ensures that software written for Plu*Perfect's DateStamperTM also runs under ZSDOS. Programs specially adapted to ZSDOS should, however, use the function calls from ZSDOS to access the clock and date stamp instead of the old DateStamperTM method.

```
GETCPV: LD      C,12
        CALL    BDOS            ; Get CP/M version number
        ...
```

```
+-----------------------------------------------------------+
|               Function 13 - Reset All Drives              |
+-------------------------------+---------------------------+
| Input:                        | Output:                   |
|   none                        |   A = 0, no file named $*.*   |
|                               |   A = 0FFH, file named $*.*   |
|                               |       available           |
+-------------------------------+---------------------------+
```

Function 13 logs out all drives and resets the address of the DMA buffer to 80H. Drive A is set as the default drive. The current user area is not changed. Before calling this function, all files into which data has been written must be closed.

The CCP uses an undocumented property of CP/M 2.2 to process submit files. Every time a drive is reset or selected under CP/M, register A contains the value 0FFH, provided a file named $*.* is available on the drive in the current user area. This returned value is used by the CCP. It shows where the $$$.SUB file could be located. The command processor then obtains the next input line not from the user, but from this file.

Several DOS systems that skip logging in from hard disks have difficulty passing this flag correctly to the CCP. ZSDOS checks for the presence of a file called $*.* each time it logs in and when files are created and deleted. The flag is set when a file named $*.* os created or discovered when logging in (to any user area). The flag is reset when the $*.* file has been successfully deleted. With this procedure, the submit flag always correctly reflects the presence of a $*.* file in the system - even if fast re-login is activated.

Since this method does not fully correspond to CP/M 2.2, the CCP can only function properly as long as there are not several files named $*.* in the system (actually very unlikely!).

In contrast to function 13, which logs out all drives, function 37 works in a more differentiated manner. This will only reset selected drives. Programs should therefore use function 37 instead of 13 if possible. Because changed disks are logged in automatically, it is rarely necessary to reset drives or to distinguish between fixed and removable disks when working under ZSDOS.

```
RESSYS: LD      C,13
        CALL    BDOS            ; Running reset, A: log in
        AND     A               ; Submit file available?
        JR      NZ,DOSUB        ; ... yes, so open the file
        ...
```

```
+-----------------------------------------------------------------+
|                   Function 14 - Select Drive                    |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   E = drive number             |   A = 0, no file named $*.*    |
|      (0 = A, 1 = B ..)         |   A = 0FFH, file named $*.*    |
|                                |       available                |
+--------------------------------+--------------------------------+
```

This function is used to select a default drive. The default drive is accessed if no drive is specified in the FCB for file access. If the selected drive has not yet been logged in, this is also done using function 14.

If one of the extended error modes of ZSDOS is active, the value 0FFH in the register does not necessarily indicate an error. Rather, the content of register H must be checked. If the value in H is zero, then no error has occurred, but ZSDOS indicates that a submit file could be present on the drive. If the value in register H is not zero, then there was a problem with the selection of the drive (generally this means: drive does not exist!).

If no extended error mode is active and the specified drive was not found, ZSDOS terminates the application program after the corresponding error message has been issued.

It should also be noted that CP/M 2.2 and all other compatible BDOS substitutes (with the exception of ZRDOS 1.9) have an error in this routine. If a drive that is not available was selected, the BDOS still assumes that the unavailable drive is the default drive. This error does not appear in normal CP/M 2.2 systems, since the BIOS reloads the CCP and the first BDOS call made represents a separate selection. NZCOM uses the BDOS to reload the CCP, which causes the error to appear.

```
; this source text assumes that a
; extended error mode is ACTIVE (return BDOS error)

SELDK:  LD      E,A             ; suppose A contains drive no.
        LD      C,14
        CALL    BDOS            ; Select drive
        AND     A
        RET     Z               ; back if no error occurred
        LD      A,H
        AND     A               ; Does 0FFH stand for submit file?
        RET     Z               ; was a submit file, Lw. OK
        ...                     ; otherwise drive not allowed
```

```
+----------------------------------------------------------------+
|               Function 15 - Open Existing File                 |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   DE = address of the FCB     |   A = directory code           |
+-------------------------------+--------------------------------+
```

Compared to other Z80 BDOS systems, ZSDOS offers a greatly expanded function for opening files. As mentioned earlier, ZSDOS can use the public file attribute and the path to open the file (both for reading and writing). If a file was opened successfully, FCB+13 contains the user area number, ORed with 80H:

```
FOPEN:  LD      A,(USER)
        LD      E,A             ; User area of the file
        LD      C,32            ; Function call set user area
        CALL    BDOS
        LD      DE,FCB
        LD      C,15
        CALL    BDOS            ; open file
        INC     A
        JP      Z,ERROR         ; Error when opening the file
        ...                     ; FCB+13 = user area
                                ; ... OR linked to 80H
```

In contrast to some other ZSDOS functions, this does not unconditionally accept the user area number in FCB+13. Digital Research originally declared the S1 byte on FCB+13 as "reserved for the system". However, it was not specified whether the byte must be set to a certain value in order to open a file. Because many programs reuse FCBs, this field can contain a valid number (and often it is), but for the former file! After many experiments, we decided that the safest way would be if ZSDOS ignored the value on FCB+13 when opening the file, if the error mode of the BDOS was zero. This is the only reliable way to support the user area number in the FCB and still remain backwards compatible.

Programs that are written to work under ZSDOS should initialize all bytes from FCB+0 to FCB+0EH. In addition to the drive and file name entries, the value of the S1 byte must be set to zero or the user area number ORed with 80H. In addition, the bytes of the current extent (FCB+12), the data module (FCB+14) and the current record (FCB+32) must be set to zero, unless the file should not be opened at the beginning. It is possible to open a file with any extent of the first data module (first 512 kBytes). However, a non-zero value cannot be specified for the data module.

If you want to use the S1 byte in an application to determine the user area when opening a file (or another file-related function), then the BDOS error mode must be used to indicate that the application knows the conditions of ZSDOS. Only then can the user area number be transferred in FCB+13. If the error mode was set to a value not equal to zero, ZSDOS uses the field FCB+13 (see function 45).

After a file has been successfully opened with function 15, the S1 byte is either set to the user area ORed with 80H or is unchanged if the error mode was zero. The data module number is always set to zero. The file name, the record counter (FCB+15) and the allocation vector (FCB+16..31) are copied from the directory entry of the corresponding file. The fields of extent, current record and optional record are retained.

For some applications it is important whether the file was opened via path or public access. ZSDOS also provides this information. After opening the attribute bit f7 is set if the path or the attribute public file were used. The code could look something like this to branch out in the case of path or public access:

```
        LD      C,15
        CALL    BDOS            ; open file
        INC     A
        JP      Z,ERROR         ; ... jump when an error occurs
        LD      HL,FCB+7        ; Pointer to the f7 bit
        BIT     7,(HL)          ; Path/public access test
        JR      NZ,ISPS         ; ... jump when used
        ...
```

A bad programming habit is to move the FCB of a file that is already open, or to use the same FCB multiple times to open additional files, as long as one file was not closed before the next was opened. Poor programming practices can cause problems with various popular operating system extensions such as BGii.

Function 16 - Close Output File	
Input: DE = address of the FCB	Output: A = directory code

With this function, all internal buffers are transferred to disk and the directory is updated when a file has been written to. A good programming style also includes closing files with function 16 if they were only open for reading. This is the only way to guarantee full compatibility with future versions of ZSDOS and other operating system extensions.

```
FCLOSE: LD      DE,FCB          ; Pointer to the file to be closed
        LD      C,16
        CALL    BDOS            ; Close file
        INC     A               ; everything OK?
        JR      Z,ERROR         ; ... jump when an error occurs
        ...
```

```
+----------------------------------------------------------------+
|             Function 17 - Search for First Entry               |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   DE = address of the FCB     |   A = directory code           |
+-------------------------------+--------------------------------+
```

This function returns the first occurrence of a matching directory entry. The match is based on the first 13 bytes of the FCB (drive, name, and extent number) as well as on the user area number (either by default or via the content of FCB+13) and the data module number (FCB+14) if a '?' is entered there. The search is always started at the beginning of the directory. Wildcards are allowed in the first 13 bytes of the FCB (FCB+0..12) and in the data module number.

Question marks are used in two ways by the Search for First Entry and Search for Next Entry functions. First, a '?' can be used in bytes FCB+1 to FCB+14 to match any character for this position. For example, if you enter a question mark in the first byte of the file name (FCB+1), all files will be found regardless of the first character (only the other characters in the file name are decisive). If you use question marks in the bytes for extent and data module, all physical extents of the file or files are found accordingly. Programs that calculate the file size by summing all extents set these two bytes to '?'.

For the second way of using question marks in the FCB, the byte for the drive (FCB+0) is assigned a '?'. In this case, however, not all drives are selected (as you might think), but all directory entries of the default drive (including entries that have already been deleted).

After calling function 17, the matching directory record is copied into the current DMA buffer. If the directory code returned in register A is shifted 5 places to the left and added to the base address of the DMA buffer, it points to the first byte of the matching directory entry.

```
SEARCF: LD      DE,DMAADDR      ; DMA address on own buffer
        LD      C,26
        CALL    BDOS
        LD      DE,FCB          ; seek this match
        LD      C,17
        CALL    BDOS            ; search for matches
        CP      0FFH            , Did it work?
        JR      Z,NOMAT         ; ... jump if there is no match
        ADD     A,A
        ADD     A,A
        ADD     A,A
        ADD     A,A
        ADD     A,A             ; Multiply index by 32
        LD      L,A
        LD      H,0             ; Make word value out of it
        LD      DE,DMAADDR      ; Address of the buffer
        ADD     HL,DE           ; HL points to agreement entry
        ...
```

Function 18 - Search for Next Entry	
Input:	Output:
none	A = directory code

After a successful search for the first entry (function 17), this function is used to search for further matches for the specified FCB (one for each call). For this function to work correctly, two conditions must be met: 1) The "Search for First Entry" function must have been executed for the first match. 2) No other BDOS calls that perform disk operations may be made between the Search for First Entry and Search for Next Entry calls.

With two exceptions, the call and return sequences correspond to those of function 17. Function 18 does not require a pointer to the FCB, since it was saved internally by DOS after the search for the first entry and is reused. Furthermore, register C must be loaded with the value 18 instead of 17 in order to select the "Search for Next Entry" function.

Function 19 - Delete File	
Input: DE = address of the FCB	Output: A = error code

This function deletes all files from the floppy disk whose directory entry corresponds to the transferred FCB. The prerequisite for this, however, is that the disk and file(s) have read/write status and the user has all Wheel usage rights, provided the Wheel Protection attribute of the file is set. The function allows the use of wild cards.

Like CP/M, ZSDOS also identifies deleted files by setting the byte of the user area of the files (DIR+0) to 0E5H and deleting the bits of the files in the allocation vector. This procedure releases the directory entries for ZSDOS, but neither the data is deleted nor the directory assignment vector (DIR+16) is changed. As long as no write operations are carried out, it is therefore often possible to undo the deletion process ("unerase"). To do this, all the physical extents of the file need to be searched for and their 0E5H changed to a permissible user area number. Function 37 is then called to cause the allocation vector to be revised.

```
FKILL:  LD      DE,FCB          ; what should be deleted
        LD      C,19
        CALL    BDOS            ; Execute delete
        INC     A               ; everything OK?
        JR      Z,ERROR         ; ... jump in case of problems
        ...
```

Function 20 - Sequential Read

Input:	Output:
DE = address of the FCB	A = read/write code

After a file with function 15 has been opened, this function can be used to transfer the next 128-byte record to the current DMA buffer. In the FCB, the entries for the current record, the current extent and the current data module number are revised to provide the next position for sequential access. By calling function 20 again, the next record can be read without intervention by the application program.

```
READS:  LD      DE,FCB          ; File must already be open
        LD      C,20
        CALL    BDOS            ; Read next record in DMA buffer
        AND     A               ; Error occurred?
        JR      NZ,ERROR        ; ... jump if there were problems
        ...
```

```
+--------------------------------------------------------------+
|                Function 21 - Sequential Write                |
+-------------------------------+------------------------------+
| Input:                        | Output:                      |
|   DE = address of the FCB     |   A = read/write code        |
+-------------------------------+------------------------------+
```

This function is the counterpart to function 20 - it writes the content of the current DMA buffer on disk to the specified file that was previously opened with function 11. In the FCB, the entries for the current record, the current extent and the data module number are revised to provide the next position for sequential access. New assignment blocks and new extents are opened or created as necessary.

With each write access to a file via the function call 21, the "archive" attribute bit (t3) of the first extent is cleared when the file is closed via function 16. A targeted backup with programs such as COPY, ZFILER, PPIP, DATSWEEP and with all other programs that support the archive attribute is possible via the archive attribute.

```
WRITS:  LD      DE,FCB          ; File must already be open
        LD      C,21
        CALL    BDOS            ; Write DMA buffer to disk.
        AND     A               ; Error occurred?
        JR      NZ,ERROR        ; ... jump if there were problems
        ...
```

Because the BDOS sets the BIOS allocation vector of a data block before sequential or random writing, it can happen that the FCB assumes an inadmissible state if a "disk full" error occurs (code 02 returned). Therefore, function 16 must always be used to close the file, which indicates errors that have occurred. Only then can further BDOS operations (e.g. deleting the faulty file) be carried out. The updated FCB is used to match the BIOS allocation vector, whereas the corresponding bits of the vector are reset after the erased file has been deleted.

```
+---------------------------------------------------------------+
|                  Function 22 - Make New File                  |
+-------------------------------+-------------------------------+
| Input:                        | Output:                       |
|   DE = address of the FCB     |   A = directory code          |
+-------------------------------+-------------------------------+
```

This function creates a new file with the name specified in the FCB on disk. The call does not occupy any data storage space on the floppy disk, but reserves the space for the first extent of the file in the directory. By using function 22, the file is opened for reading and writing at the same time, so that function 15 no longer has to be called up separately.

WARNING: The "Make New File" function does not check whether a file with the same name already exists on the disk before it is created. The programmer must ensure that an existing file name is not used again for the creation of the file.

Example:

```
FMAKE:  LD      DE,FCB          ; this file is to be created
        LD      C,17            ; secure against duplicates first
        CALL    BDOS
        INC     A
        JR      Z,FMAKE1        ; ... not yet available - create
        LD      DE,DUPWRN       ; Alert users to their presence
        LD      C,9
        CALL    BDOS
        LD      C,1
        CALL    BDOS            ; what does the user want to do?
        AND     5FH             ; convert to uppercase
        CP      'Y'
        LD      C,0
        CALL    NZ,BDOS         ; ... if NO, then termination
        LD      DE,FCB
        LD      C,19
        CALL    BDOS            ; otherwise delete duplicate
FMAKE1: LD      DE,FCB
        LD      C,22
        CALL    BDOS            ; now the file is created
        INC     A               ; Error occurred?
        JR      Z,ERROR         ; ... jump if there were problems
        ...
DUPWRN: DEFB    'File exists! Erase it (Y/N)? $'
        ...
```

```
+----------------------------------------------------------------+
|                   Function 23 - Rename File                     |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   DE = address of the FCB      |   A = error code              |
+--------------------------------+-------------------------------+
```

This function is used to rename the file that is named with the first 13 bytes of the FCB (including the drive and optional user area information). The new file name is transferred from byte 17 of the FCB. In contrast to CP/M 2.2 and ZRDOS, ZSDOS allows the specification of wild cards for this function call. If there is a question mark '?' at any point in the original file name, this character is not changed in the directory by the rename function. All attributes with the exception of "public file" are retained. For security reasons, the renamed files are all given the "private" attribute to prevent conflicts due to multiple public files of the same name on a floppy disk.

As with the "Create file" function, the programmer is responsible for avoiding duplicates in the directory with this function.

FCB format when renaming:

```
Offset: 0  1     9   12 13 14 15 16 17    25  28 29 30 31
        |  |     |   |  |  |  |  |  |     |   |  |  |  |
Data:   dr oldfn typ 0  us 0  0  0  newfn typ 0  0  0  0
```

```
Abbreviations:  dr - drive
                oldfn - old filename
                typ - filetype
                us - user
                newfn - new filename
```

```
RENAME:                         ; Prereq: name not yet assigned
        LD      DE,RENFCB       ; appropriately formatted FCB
        LD      C,23
        CALL    BDOS            ; Rename file
        INC     A               ; any problems?
        JR      Z,ERROR         ; ... jump when an error occurs
        ...
```

```
+-----------------------------------------------------------------+
|              Function 24 - Get Active Drive Map                 |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   none                         |   HL = login vector            |
+--------------------------------+--------------------------------+
```

This function returns a bit map of the currently logged in drives in register pair HL. The assignment is defined as follows:

```
Register:       H                       L
Bit:            7 6 5 4 3 2 1 0         7 6 5 4 3 2 1 0
Drive:          P O N M L K J I         H G F E D C B A
```

```
GETLGV: LD      C,24            ; Get login vector
        CALL    BDOS            ; Vector now in HL
        ...
```

Function 25 - Get Default Drive Number

Input:	Output:
none	A = default drive

This function can be used to determine the default drive (the drive that is used without specifying a drive number). A value between 0 and 15 is returned in register A, which designates the corresponding drive from A to P. For example, the value 0 means drive A and 2 means drive C.

```
SHOWDD: LD      C,25            ; get default drive
        CALL    BDOS
        ADD     A,41H           ; Convert to ASCII
        LD      C,2             ; output to console
        LD      E,A             ; for ZSDOS load in E.
        CALL    BDOS            ; output as ASCII on CON:.
        ...
```

```
+-----------------------------------------------------------------+
|              Function 26 - Set File Buffer Address              |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   DE = DMA address             |   none (A = 00H)               |
+--------------------------------+--------------------------------+
```

This function sets the address of the 128 byte buffer, which is used for disk write and read functions as well as for the transfer of the date stamp, to the address given in register pair DE. By default, the DMA buffer starts at address 80H when a program is started. The current address can be determined using function 47.

Note that the term DMA is not entirely correct. The abbreviation DMA stands for "Direct Memory Access", which is actually a peripheral chip. Depending on the hardware and the BIOS of the respective system, such a circuit may or may not be used for the transmission of the disk data.

```
STDMA:  LD      DE,DMAADR       ; Data transfers take place there
        LD      C,26
        CALL    BDOS
        ...
```

Function 27 - Get Allocation Vector

Input:	Output:
none	HL = address of allocation vector

This function returns the address of the allocation vector for the default drive in the HL register pair.

```
GETALV: LD      E,0             ; Get allocation vector
        LD      C,14            ; from drive A
        CALL    BDOS            ; Set A as the default drive
        LD      C,27            ; get allocation vector now
        CALL    BDOS
        ...
```

Function 28 - Write Protect Drives

Input:	Output:
DE = write protection vector	none (A = 00H)

This function is primarily used to write protect the disks in the selected drives from attempts to write, rename, delete, etc. A bit map for determining the drives is transferred in register pair DE.

The possibility of resetting drives to read/write at a later time depends on the status of the "Read-only vector received" flag bit in the configuration byte (bit 2 of the memory cell at address ZSDOS BASE+15). If the bit is set, the write protection vector is never cleared. If the bit is cleared, the respective drive is set to the read/write status by calling functions 13 or 37.

A drive is assigned to each bit in register pair DE:

Register:	D								E							
Bit:	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Drive:	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

```
SETDRO: LD      C,25            ; get default drive
        CALL    BDOS
        LD      HL,1            ; Initialize the mask in HL
        AND     A               ; Test for drive A.
        JR      Z,SETDR2        ; ... jump if drive is A
        LD      B,A             ; otherwise value = shift counter
SETDR1: ADD     HL,HL           ; Shift 16 bits to the left
        DJNZ    SETDR1          ; ... repeat until done
SETDR2: EX      DE,HL           ; Vector in DE
        LD      C,28            ; set default drive to R/O
        CALL    BDOS
        ...
```

Function 29 - Get Read-Only Map

Input:	Output:
none	HL = read-only map vector

This function returns an image of the drives that were write-protected with function call 28. In contrast to CP/M, drives are not automatically write-protected when a floppy disk change is detected. The format of the write protection vector that is returned in register pair HL is defined as follows:

Register:	D	E
Bit:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Drive:	P O N M L K J I	H G F E D C B A

```
CHKRO:  LD      C,25            ; get default drive
        CALL    BDOS
        LD      HL,1            ; Initialize the mask in HL
        AND     A               ; Test for drive A.
        JR      Z,CHKRO2        ; ... jump if A is default
        LD      B,A             ; otherwise value = shift counter
CHKRO1: ADD     HL,HL           ; Shift 16 bits to the left
        DJNZ    CHKRO1          ; ... repeat until done
CHKRO2: PUSH    HL              ; Secure mask
        LD      C,29            ; get write protection vector
        CALL    BDOS
        POP     DE              ; Restore mask
        LD      A,E
        AND     L
        LD      L,A             ; Match test by
                                ; AND operation of the LSBs
        LD      A,D
        AND     H
        LD      H,A             ; ... and MSBs
        OR      L               ; Check for agreement
        JR      NZ,ISWP         ; if not zero then
                                ; Standard drive read-only
        ...

; Routine to reset write protection for all drives

RST2RW: LD      C,100           ; check the flags,
        CALL    BDOS            ; ... for "R/O vector received"
                                ; is active
        BIT     2,L
        JR      Z,RESET         ; deactivated, reset works
        RES     2,L             ; otherwise deactivate first
        EX      DE,HL
        LD      C,101
        CALL    BDOS            ; Set flags
RESET:  LD      C,29
        CALL    BDOS            ; check for a drive
```

```
                                ; is write protected
        LD      A,H
        OR      L
        RET     Z               ; no, so omit resetting
        EX      DE,HL           ; Write protection vector in DE
        LD      C,37            ; reset multiple drives
        CALL    BDOS            ; ... to remove write protection
        ...
```

Function 30 - Set File Attributes

Input:	Output:
DE = address of the FCB	A = error code

File attributes are used by ZSDOS to control the status of files between BDOS calls. The attributes are contained in the most significant bits (bit 7) of the file names (8 bytes name, 3 bytes type) of the FCB and directory entry. The following meanings are defined:

FCB+1	f1 (available for users; attribute "do not load" with Plu*Perfect)
FCB+2	public file
FCB+3	no access stamp
FCB+4	f4 (available for users)
FCB+5	reserved for internal use by ZSDOS
FCB+6	reserved for internal use by ZSDOS
FCB+7	reserved for internal use by ZSDOS
FCB+8	wheel protection
FCB+9	Read only (write protection)
FCB+10	system file
FCB+11	archived

This function allows the programmer to set or clear attributes by setting or resetting the corresponding bits in the FCB, the address of which is transferred in DE. Wildcards may appear in the FCB.

The user must not change the attribute bits that are reserved for internal use. These bits were already reserved under CP/M and ZRDOS, so this is not a new restriction.

If the attribute bit for public files is set, this file can be found by any user area on the same floppy disk if a unique file specification is used for the search. It is the responsibility of the programmer to ensure that there is no second file on a disk that has the same name as a public file.

The wheel protection attribute prevents the file from being overwritten, deleted, renamed or one of its attributes changed as long as the wheel byte is not on. In a system environment without ZCPR, this attribute generally has no effect. ZSDOS then assumes that the user has all rights of use.

The read-only attribute provides full protection against overwriting, deleting or renaming a file.

```
SETATT: LD      DE,FCB          ; FCB has attributes to be set
        LD      C,30            ; ... or reset
        CALL    BDOS            ; Set file attributes
        ...
```

Function 31 - Get Disk Parameters

Input:	Output:
none	HL = address of the DPB

This function returns the address of the disk parameter block (DPB) for the default drive in the HL pair of registers. Information such as the capacity of the drive, the number of directory entries, the number of tracks, etc. can be obtained.

```
GETDPB: LD      E,0             ; Drive A DPB
        LD      C,14
        CALL    BDOS            ; The default drive is now A
        LD      C,31            ; get DPB
        CALL    BDOS            ; Pointer to DPB from A: in HL
        ...
```

Function 32 - Get or Set User Area	
Input:	Output:
E = 0FFH (get)	A = user area
E = 0-31 (set)	A = 0

This function gets or sets the default user area for file operations. Compared to CP/M and ZRDOS, ZSDOS allows the default user area to be overwritten by specifying it separately in the FCB. Otherwise the default user area is used for all file operations.

```
GETUSR: LD      C,32
        LD      E,0FFH          ; get current user area
        CALL    BDOS
        ...


SETUSR: LD      E,A             ; assumed, user area in A
        LD      C 32
        CALL    BDOS            ; set new user area
        ...
```

```
+----------------------------------------------------------------+
|               Function 33 - Random Access Read                 |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   DE = address of the FCB      |   A = read/write code         |
+--------------------------------+-------------------------------+
```

With this function the read access to the file specified in the FCB is possible. Before calling the function, the random access record number (FCB+33..FCB+35, LSB..MSB) must be set to the number of the desired 128-byte record. Before this, the file must have been opened with extent 0 using function 15.

While the sequential read function increments the current record (FCB+32) after each access, function 33 leaves it unchanged. The random access record number must be set by the application program before each call.

```
RDRAN:  ...                     ; FCB+33...35 already set
        LD      DE,FCB          ; File has already been opened
        LD      C,33
        CALL    BDOS            ; read record into the DMA buffer
        AND     A               ; Error occurred?
        JR      NZ,ERROR        ; ... jump if there were problems
        ...
```

```
+----------------------------------------------------------------+
|                Function 34 - Random Access Write               |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   DE = address of the FCB      |   A = read/write code         |
+--------------------------------+-------------------------------+
```

This function allows random write access to the file specified in the FCB. As with function 33, the random access record number (FCB+33..FCB+35, LSB..MSB) must be set to the number of the desired 128-byte record. The file must have already been opened with function 15 or created with function 22. Please note that when opening an existing file with function 15, the extent number must be 0 (the base extent).

This function also does not affect the value for the current record (FCB+32), so that the application program must set the number for the optional record before each function call.

This function opens, closes and creates extents as needed. The function also causes the archive bit of the first extent to be reset using function 16 when the file is closed. This indicates that the file has been changed.

Remarks:

1.) Under ZSDOS it is possible to create files that are too large to work under CP/M 2.2 or ZRDOS. While only up to 65,536 records (8,192 kB) can be processed by CP/M 2.2 and ZRDOS files, ZSDOS supports files up to a size of 262,144 records (32,768 kB). Such large files can easily be used under CP/M 3.0.

2.) Files that were created randomly and contain "holes" are not transmitted correctly by most copying programs, since they carry out sequential read and write operations. These programs include, for example, COPY, PIP and PPIP.

```
WRRAN:  ...                     ; FCB+33...35 already set
        LD      DE,FCB          ; File has already been opened
        LD      C,34
        CALL    BDOS            ; write record from DMA to disk
        AND     A               ; Error occurred?
        JR      NZ,ERROR        ; ... jump if any problems
        ...
```

If an "Disk full" error occurs, the same process applies as for function 21: The file should be closed before performing other operations (see also section 5.2.21).

```
+----------------------------------------------------------------+
|              Function 35 - Get File End Address                |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   DE = address of the FCB     |   A = error code               |
|                               |   FCB+33..35 = last record+1   |
+-------------------------------+--------------------------------+
```

This function calculates the "virtual" size of a file. The returned size is determined by setting the random record number in the FCB to that of the last record found plus one. Do not confuse this value with the real size of the file on the disk - files created randomly can contain "empty" extents, which do not take up space on the disk, but are included in the calculation of this function.

This function call is often used to set the number of the random record to a value after the end of the file when adding further records.

```
GETSIZ: LD      DE,FCB          ; Get the size of this file
        LD      C,35
        CALL    BDOS            ; Read size in FCB+33...35
        AND     A               ; Error occurred?
        JR      NZ,ERROR        ; ... jump if there were problems
        ...
```

```
+----------------------------------------------------------------+
|               Function 36 - Get Random Address                 |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|    DE = address of the FCB    |   A = 00H                      |
|                               |   FCB+33..35 = current record  |
+-------------------------------+--------------------------------+
```

This function sets the random record number in the FCB to the current position in the file, which is accessed sequentially for reading and/or writing. This information can be saved for later use to quickly access the desired location in the file.

A possible application would be the creation of an index during the sequential writing of a file. Using the index, the indexed positions can be accessed quickly with random accesses. Please note that sequential read and write functions do not change the random record number.

```
SETRAN: LD      DE,FCB          ; File is already open
        LD      C,36            ; ... and accessed sequentially
        CALL    BDOS            ; set FCB+33..FCB+35
        ...                     ; ... to current record number
```

Function 37 - Reset Drives

Input:	Output:
DE = mask	A = 0 or A = 0FFH if file named $*.* exists on current disk

Under CP/M 2.2, this incorrectly programmed function could hardly be used to log out several drives. Most programmers assumed that function 37 would log in a reset drive under CP/M - but that was not the case. Most BDOS substitutes adopted this bug, but it has been fixed in ZSDOS. If the default drive is reset, it is then logged in again and the allocation vector is rebuilt - even if it is a hard disk. Before doing this, you should make sure that all files on the disk are closed before the drive is reset with function 37.

Note: If a program contains a place to change the floppy disk, the function should only be carried out afterwards.

Function 37 offers the only available method for re-building the allocation vectors of fixed disks when "fast re-login of fixed disks" is enabled. Furthermore, this function must always be carried out when direct BIOS calls have been used by an application program that change directories or allocation vectors from fixed disks.

A drive is assigned to each bit in register pair DE as follows:

Register:	D								E							
Bit:	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Drive:	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Here is an example of how to log all fixed disks back into the system:

```
        LD      C,39            ; get vector fixed disks
        CALL    BDOS
        EX      DE,HL           ; Vector transferred in DE
        LD      C,37
        CALL    BDOS            ; log in fixed disks again
        ...
```

```
+-----------------------------------------------------------------+
|            Function 39 - Get Vector of Fixed Disks              |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   none                         |   HL = vector of fixed disks   |
+--------------------------------+--------------------------------+
```

Function 39 reproduces a bit image of the drives in the system that are logged in as fixed disks when "fast re-login of fixed disks" is enabled. An example of the use of this function call can be found in the listing for function 37 in this section.

The bit map, which is returned by ZSDOS in the HL register pair, is defined as follows:

```
Register:       H                       L
Bit:            7 6 5 4 3 2 1 0         7 6 5 4 3 2 1 0
Drive:          P O N M L K J I         H G F E D C B A
```

```
+----------------------------------------------------------------+
|       Function 40 - Random Access Write with Zero Fill         |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   DE = address of the FCB      |   A = read/write code         |
+--------------------------------+-------------------------------+
```

The function of this function is very similar to function 34, however, all records of a newly occupied block are initialized by filling with 00H. If records are created using function 34 in blocks that have never been written to, undefined data may be contained in the blocks.

```
WRRANZ: ...                     ; FCB+33...35 already set
        LD      DE,FCB          ; File has already been opened
        LD      C,40
        CALL    BDOS            ; writes record from DMA to disk
                                ; ... fill the rest with zeros
        AND     A               ; Error occurred?
        JR      NZ,ERROR        ; ... jump if any problems
        ...
```

```
+-----------------------------------------------------------------+
|               Function 45 - Set BDOS Error Mode                 |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   E = xxxxxxx1B: suppress the  |   none                         |
|       error messages           |                                |
|   E = xxxxxx1xB: return error  |                                |
|       code to program          |                                |
|   E = 1xxxxx00B: set ZSDOS     |                                |
|       default error mode;      |                                |
|       display error message    |                                |
|   E = 00000000B: set CP/M      |                                |
|       default error mode;      |                                |
|       display error message    |                                |
+--------------------------------+--------------------------------+
```

Function 45 enables application programs to influence the error handling of ZSDOS. All errors detected by the BDOS, including select and write protect errors, can be transferred to the application program and an additional screen message from ZSDOS can be displayed or suppressed.

In order to define the error mode, the value is set in register E when function 45 is called. If the value in register E has bit 0 reset, when an error occurs a message is output on the screen by ZSDOS. If bit 0 is set, no message is output.

If bit 1 is register E is set, when an error occurs control is passed back to the program. Register A then contains the value 0FFH and the extended error code is available in register H.

If register E is set to 0 when setting the error mode, ZSDOS sets the default CP/M error mode. These values of the input parameters correspond to those of function 45 under CP/M Plus.

The value passed to function 45 serves another purpose - it informs the DOS that a ZSDOS program is running. Programs written especially for ZSDOS set the S1 byte in the FCB to the user area ORed with 80H and the S2 byte to the value 0 when file operations such as opening, renaming, deleting etc. are carried out. If the error mode has been set to a non-zero value, ZSDOS assumes that the application program is setting these bytes correctly.

In order to inform the DOS that a ZSDOS program is running while maintaining the default error mode, bit 7 in register E is used as a flag during the transfer. The error mode bits are currently set as follows:

```
Bit: 7 6 5 4 3 2 1 0
     | | | | | | | +- Display suppression flag
     | | | | | | +--- Error code return flag
     | +-+-+-+-+----- reserved
     +--------------- Flag for information "ZSDOS program"
```

If an application program has changed the error mode, it must be reset to zero before returning to the operating system level. This is the only way to ensure full compatibility with programs that were not specifically written for ZSDOS. If a program is terminated from the BDOS, the error mode is set to 0 before booting. Calling function 0 also resets the error mode to the CP/M default.

When extended error codes are returned in the corresponding modes, the value 0FFH in register A indicates the occurrence of an error. The respective error code is returned in register H. The codes have the following meaning:

Value in H	Meaning
0	no extended error code
1	disk I/O error (bad sector)
2	disk is write-protected
3	file is read-only
4	illegal drive specification

Only 4 extended error codes are currently defined. However, it is possible that more will be added in future versions. For this reason, it should not be assumed during programming that only certain errors can occur, but that all defined error codes are tested.

Examples for setting the failure mode:

```
SET$RET$ERR:
        LD      E,03H           ; Error code return, no error
        LD      C,45             ; message displayed
        CALL    BDOS
        ...


SET$QRET$ERR:
        LD      E,02H           ; Error code return, display
        LD      C,45             ; error message
        CALL    BDOS
        ...


SET$DEF$ERR:
        LD      E,0              ; Set default error mode
        LD      C,45
        CALL    BDOS
        ...
```

Examples of error handling:

```
FOPEN:  LD      DE,FCB
        LD      C,15
        CALL    BDOS             ; open file
        INC     A
        JR      NZ,OKOPEN       ; no error, file is open
        LD      A,H             ; Load extended error code
        AND     A               ; Extended error code test
        JR      Z,NOFILE        ; not an advanced bug
                                ; ...File was not found
        CP      1
        JR      Z,BADSEC        ; Bad sector error
;
; Write protection errors cannot occur when opening
;
        CP      4
        JR      Z,SELERR        ; "Illegal drive" error
;
; These extended error codes have been defined so far.
; There may be more in future versions, so
; you should plan this possibility!
;
        JR      UNKERR          ; ... else jump to routine
                                ; "unknown error"
        ...


FWRITE: LD      DE,FCB
        LD      C,21
        CALL    BDOS            ; Write sector to file
        AND     A
        JR      Z,OKWR          ; no error, file is open
        CP      0FFH            ; Is it an advanced bug?
        JR      NZ,NRMERR       ; No, normal mistake
                                ; ... like "disk full" etc.
        LD      A,H             ; else expanded error code
        CP      1
        JR      Z,BADSEC        ; ... "bad sector"
        CP      2
        JR      Z,DISKWP        ; ... "Disk R/O"
        CP      3
        JR      Z,FILEWP        ; ... "File R/O"
        CP      4
        JR      Z,SELERR        ; ... "illegal disk"
;
; These extended error codes have been defined so far.
; There may be more in future versions, so
; you should plan this possibility!
;
        JR      UNKERR          ; ... else jump to routine
                                ; "unknown error"
        ...
```

```
+-----------------------------------------------------------------+
|              Function 47 - Get File Buffer Address              |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   none                         |   HL = pointer to DMA buffer   |
+--------------------------------+--------------------------------+
```

This function returns the address of the DMA buffer in register pair HL. The DMA buffer is used for all transfers from and to the disk as well as for the date stamp functions. Mainly IOPs that re-enter ZSDOS use this function. The address of the DMA buffer is fetched from the IOP in order to reset it to the old value after the call comes back.

It should be noted that only the DMA address is returned as it is known to the DOS. The DMA address in the BIOS can differ if an application program has changed it through direct BIOS calls. In order to remain compatible with later operating systems, the use of BIOS routines should be minimized or completely avoided.

```
GETDMA: LD      C,47            ; get address of the DMA buffer
        CALL    BDOS            ; is returned in HL
        ...
```

Function 48 - Get BDOS Version Number	
Input: none	Output: H = BDOS ID ('S' for ZSDOS) ('D' for ZDDOS) A, L = version number

With this function you can find out whether ZSDOS is running in a system and if so, which version. It is absolutely necessary to determine the presence of ZSDOS before using one of the new or expanded ZSDOS functions. To ensure that ZSDOS is available, first call function 12, from which 22H must be returned in register A. Function 48 is then called.

The function for querying the labeling of extended operating systems was developed in collaboration with Joe Wright, Bridger Mitchell and the authors of ZSDOS. The call is identical to the ZRDOS function "Get version number". The DOS substitutes can be distinguished on the basis of the indicator returned in register H. An extended DOS can be recognized by the fact that, in contrast to CP/M, the version number is returned in registers A and L. This function is not included in the normal CP/M BDOS and only the value zero is returned. The following indicators are currently assigned to the DOS:

H Value	DOS
00H	ZRDOS
'D'	ZDDOS
'S'	ZSDOS

The assignment of new values should be coordinated with one of the above-mentioned persons in order to avoid any misunderstandings regarding this function. Of the 256 different values, 253 are still available!

```
        LD      C,12            ; get CP/M version
        CALL    BDOS
        CP      22H             ; compatible with version 2.2?
        JR      NZ,NOTZS        ; ... no, so it can't be ZSDOS
        LD      C,48
        CALL    BDOS            ; get version of the extended DOS
        LD      A,H             ; Load version indicator
        CP      'S'             ; Is it ZSDOS?
        JR      NZ,NOTZS        ; ... jump if not ZSDOS
        ...
```

```
+----------------------------------------------------------------+
|                 Function 98 - Get System Time                  |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   DE = time info block address|   A = time/date code           |
+-------------------------------+--------------------------------+
```

This function enables application programs to read the system clock. In the register pair DE, a target address of a buffer area is transferred, in which the time information from ZSDOS is written.

Driver routines are required for this function. If no driver is installed or the clock cannot be read, the value 0FFH is returned in register A and the buffer remains unchanged. If register A contains the value 1 on return, the time and date information of the system clock is available in the buffer.

```
GETTIM: LD      DE,TIMEAD       ; Start address of the buffer
        LD      C,98
        CALL    BDOS            ; get time information in buffer
        INC     A               ; Error occurred?
        JR      Z,ERROR         ; ... jump if not available
        ...

TIMEAD: DEFB    0,0,0,0,0,0     ; Buffer initialized with zeros
        ...
```

```
+-----------------------------------------------------------------+
|                  Function 99 - Set System Time                  |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   DE = address of time block   |   A = time/date code           |
+--------------------------------+--------------------------------+
```

With this function, an application program can set the system clock. The start address of the buffer with the time information is transferred in register pair DE.

Driver routines are required for this function. If no driver is installed, the value 0FFH is returned in register A. If register A contains the value 1 on return, the clock has been set.

```
SETTIM: LD      DE,TIMEAD       ; Start address of the buffer
        LD      C,99
        CALL    BDOS            ; Set the clock
        INC     A               ; Error occurred?
        JR      Z,ERROR         ; ... jump, in case of errors or
                                ; ... if there is no function
        ...
```

```
+----------------------------------------------------------------+
|            Function 100 - Get Configuration Flags              |
+--------------------------------+-------------------------------+
| Input:                         | Output:                       |
|   none                         |   HL = flags                  |
+--------------------------------+-------------------------------+
```

This function returns the current ZSDOS configuration flags in the HL register pair. In version 1.1 of ZSDOS, only register L is important. For reasons of compatibility with later versions of ZSDOS, which may use more status bits, the value 0 is returned in register H.

Register H = 0, register L contains:

```
Bit: 7 6 5 4 3 2 1 0
     | | | | | | | +- Public files              on (1)/off (0)
     | | | | | | +--- Write public/path files    on (1)/off (0)
     | | | | | +----- Get read-only vector       on (1)/off (0)
     | | | | +------- quick login                on (1)/off (0)
     | | | +--------- Floppy disk change warning on (1)/off (0)
     | | +----------- ZCPR2/3 path               on (1)/off (0)
     | +------------- Path with/out system files on (1)/off (0)
     +--------------- reserved
```

Please refer to section 4.3.4 for a detailed description of the functions of the individual bits.

```
+----------------------------------------------------------------+
|            Function 101 - Set Configuration Flags              |
+-------------------------------+--------------------------------+
| Input:                        | Output:                        |
|   DE = flags                  |   None                         |
+-------------------------------+--------------------------------+
```

With this function call, the ZSDOS configuration flags are set to the values transferred in register pair DE. In version 1.1 of ZSDOS, only the value in register E is important. Register D should be loaded with the value 0 in order to remain compatible with later versions.

Register D = 0, register E contains:

```
Bit: 7 6 5 4 3 2 1 0
     | | | | | | | +- Public files               on (1)/off (0)
     | | | | | | +--- Write public/path files    on (1)/off (0)
     | | | | | +----- Get read-only vector       on (1)/off (0)
     | | | | +------- quick login                on (1)/off (0)
     | | | +--------- Floppy disk change warning on (1)/off (0)
     | | +----------- ZCPR2/3 path               on (1)/off (0)
     | +------------- Path with/out system files on (1)/off (0)
     +--------------- reserved
```

Please refer to section 4.3.4 for a detailed description of the functions of the individual bits.

Function 102 - Get Date Stamp	
Input: DE = address of the FCB	Output: A = time/date code Date stamp in the DMA buffer

This function returns the date stamp of the file whose name is passed in the FCB addressed by DE. The extent and module numbers (FCB+12...FCB+4) must be set to zero before the function is called. The correct user area must also be set. To do this, either place the user area number ORed with 80H in FCB+13 (S1) or call function 32 beforehand. The desired stamp information is available in the first 15 bytes of the DMA buffer after calling function 102. Future versions of ZSDOS may use an extended stamp format, so we recommend providing a 128-byte buffer for the date stamp.

In order to be able to carry out these functions, appropriate driver routines must be installed. If no drivers are available or the stamps cannot be read, the value 0FFH is returned in register A. In this case the content of the DMA buffer is undefined. If the value 1 is returned in register A, then valid stamp information is available in the DMA buffer.

```
+-----------------------------------------------------------------+
|                  Function 103 - Set Date Stamp                  |
+--------------------------------+--------------------------------+
| Input:                         | Output:                        |
|   DE = address of the FCB      |   A = time/date code           |
|   Date stamp in the DMA buffer|                                 |
+--------------------------------+--------------------------------+
```

This function writes the stamp information in the DMA buffer to disk. When the function is called, the register pair DE points to the FCB with the name of the file to be stamped. As with function call 102, the extent and module numbers must also be initialized and the user area defined before the function is called.

In order to be able to carry out these functions, appropriate driver routines must be installed. If no drivers are available or the stamps cannot be written, register A contains the value 0FFH when returning. If the value 1 is returned in register A, the stamp information has been successfully written to disk. Please note that ZSDOS does not call normal error handling if the disk is write protected. In this case, the value 0FFH is returned in register A and no date stamp is written to disk.

Example of using functions 102 and 103:

```
COPYDS: LD      DE,DSBUF        ; Buffer for stamp information
        LD      C,26            ; set address of the DMA buffer
        CALL    BDOS
        LD      DE,SRCFCB       ; Source FCB
                                ; (User area already set)
        LD      C,102
        CALL    BDOS            ; get stamp of the source file
        LD      DE,DSTFCB       ; Target FCB
                                ; (User area already set)
        LD      C,103
        CALL    BDOS            ; Transfer stamp to target file
        ...
```

Quick overview of the functions of ZSDOS

Nr.	Function name	Input parameters	Returned values
0	Terminate Program	none	none
1	Console Input	none	A=char
2	Console Output	E=char	none (A=BIOS A)
3	Reader Input	none	A=character
4	Punch Output	E=char	none (A=BIOS A)
5	List Output	E=char	none (A=BIOS A)
6	Direct Console I/O	E=0FFH (in)	A=entered char
		E=0FEH (in)	A=console status
		E=0FDH (in)	A=entered char
		E=0..0FCH (out)	none (A=BIOS A)
7	Get IOBYTE	none	A=IOBYTE
8	Set IOBYTE	E=IOBYTE	none (A=IOBYTE)
9	Output String	DE=string address	B none (A='$')
10	Read Buffer	DE=buffer address	none (A=0DH)

11	Get Console Status	none	A=00H - no char
			A=01H - char

12	Get CP/M Version	none	HL=22H

13	Reset All Drives	none	A=00H no $*.* File
			A=0FFH $*.* Available

14	Select Drive	E=drive number	A=00H no $*.* File
			A=0FFH $*.* Available

15	Open existing file	DE=FCB address	A=directory code
16	Close output file	DE=FCB address	A=directory code
17	Search for First	DE=FCB address	A=directory code
18	Search for Next	none	A=directory code
19	Delete File	DE=FCB address	A=error code
20	Sequential Read	DE=FCB address	A=read/write code
21	Sequential Write	DE=FCB address	A=read/write code
22	Make New File	DE=FCB address	A=directory code
23	Rename File	DE=FCB address	A=error code
24	Get Login Vector	none	HL=login vector
25	Get Default Drive	none	A=default drive
26	Set File Buffer	DE=DMA address	none (A=00H)
27	Get Allocation Vector	none	HL=allocation vector
28	Set Read-Only Vector	DE=R/O vector	none (A=00H)
29	Get Read-Only Vector	none	HL=R/O vector
30	Set File Attributes	DE=FCB address	A=error code
31	Get DPB Address	none	HL=address DPB
32	Get/Set User Code	E=0FFH (get)	A=user area
		E=user area (put)	A=00H
33	Random Access Read	DE=FCB address	A=read/write code
34	Random Access Write	DE=FCB address	A=read/write code
35	Calculate File Size	DE=FCB address	A=error code FCB+33..35=# Rec.+1
36	Set Direct Record	DE=FCB address	A=00H

37	Reset Drives	DE=mask	A=00H reset to default
			A=0FFH $*.* available

38	not included		

39	Get Fixed Disks	none	HL=fixed disks vector
40	Random Access Write with Zero Fill	DE=FCB address	A=read/write code
41	not included		
42	not included		
43	not included		
44	not included		
45	Set BDOS Error Mode	E=0FFH code	A=00H
		E=0FEH code+msg	A=00H
		E=80H ZSDOS mode	A=00H
		E=00H CP/M mode	A=00H
46	not included		
47	Get File Buffer	none	HL=pointer to DMA
48	Get BDOS Version	none	H=DOS type
			'S'=ZSDOS
			'D'=ZDDOS
			A,L=version (BCD)
98	Get System Time*	DE=time block addr	A=time/date code
99	Set System time*	DE=time block addr	A=time/date code
100	Get Config Flags	none	HL=flags
101	Set Config Flags	DE=flags	none
102	Get Date Stamp^	DE=FCB address	A=time/date code
			Stamp in the DMA
103	Set Date Stamp^	DE=FCB address	A=time/thumb code
			Stamp in the DMA

* Functions 98 and 99 are only available if a clock driver module is installed.

^ Functions 102 and 103 are only available under ZSDOS if a Date stamp module is installed.

Overview of the BDOS error codes

Directory code:
A = 00H, 01H, 02H, 03H if no error has occurred
A = 0FFH, in the event of an error

Error code:
A = 00H, no error
A = 0FFH, error occurred

Time/date code:
A = 01H if no error has occurred
A = 0FFH, error occurred

Read/write code:
A = 00H if no error has occurred
A = 01H, read - end of file
write - Directory full
A = 02H, floppy disk full
A = 03H, error while closing on random Read Write
A = 04H, empty record for random reading
A = 05H, directory full of random writing
A = 06H, record during random reading/writing too large

extended error codes in error mode:
A = 0FFH, further error codes in H
H = 01H, disk I/O error (defective sector)
H = 02H, floppy disk write-protected (read only)
H = 03H, file read-only
H = 04H, illegal drive selected

Brief overview of the BIOS functions

Number	Function name	Input parameters	returned values
0	BOOT	none	none
1	WBOOT	none	none

2	CONST	none	A=0FFH, ready
			A=00H, not ready

3	CONIN	none	A=char from CON:
4	CONOUT	C=char to CON:	none
5	LIST	C=char to LST:	none
6	PUNCH	C=char to PUN:	none
7	READER	none	A=char from RDR:
8	HOME	none	none

9	SELDSK	C=drive (0..15)
		E=Init Select Flag
		HL=0 impermissible running
			HL=address DPH

10	SETTRK	BC=track number	none
11	SETSEC	BC=sector number	none
12	SETDMA	BC=DMA buffer address	none

13	READ	no	A=00H okay
			A=01H error

14	WRITE	C=00H write data
		C=01H Directory write
		C=02H write new data
			A=00H okay
			A=01H error

15	LISTST	none	A=00H ready
			A=0FFH not ready

16	SECTRN	BC=log. sector #
		DE=translation addr
			HL=phys. sec. #

Note: The BIOS must not change the IX register!

Harold F. Bower
7914 Redglobe Court
Severn, MD 21144
Ladera Z node
213/670-9465

Cameron W. Cotrill
2935 Manhattan Ave.
La Crescenta, CA 91214
Ladera Z node
213/670-9465

Carson Wilson
1359 W. Greenleaf
Chicago, IL 60626
Antelope Freeway Z-Node
312/764-5162