MSX-DOS version 2

22nd October 2001

Contents

Chapter 1

Command specification

This chapter describes the user interface and commands provided by MSX-DOS 2 version 2.20.

1.1 Introduction

MSX-DOS 2, like its predecessor MSX-DOS 1, is provided in a cartridge and in some disk files. The disk files are `MSXDOS2.SYS`, `COMMAND2.COM`, help files and transient commands.

`MSXDOS2.SYS` has the ability to load and execute programs in an enhanced CP/M-compatible environment. `COMMAND2.COM` is a special program which, when loaded and executed, provides the user with many sophisticated commands and features generally compatible with and in many cases better than those found in MS-DOS and MSX-DOS 1, such as extended memory management.

It also has the ability to load and execute specially written MSX-DOS 1 programs and most standard CP/M programs, and can execute batch files with parameter substitution and other features similar to those found in MS-DOS.

An `APPEND` facility is provided to increase the ease of use of directories with CP/M programs which were not written to handle them.

Throughout the rest of this manual the term MSX-DOS is used to mean MSX-DOS version 2.xx unless otherwise stated.

1.2 Editing command lines

When typing in a command line to MSX-DOS, a simple editing facility is available for correction of mistakes or the re-entering and editing of previous commands.

Typing ordinary characters at the keyboard cause the characters to appear on the screen as would be expected. Typing most control characters cause them to be represented by a '^' symbol followed by the control letter. Exceptions are

carriage return (RET or CTRL-M), back space (BS or CTRL-H), tab (TAB or CTRL-I), insert (INS or CTRL-R), escape (ESC or CTRL-[), home (HOME or CTRL-K), CTRL-C, CTRL-J, CTRL-N, CTRL-P, CTRL-S, CTRL-U and CTRL-X (SELECT). These perform the following functions:

CTRL-C This acts as a 'break' key. A more drastic and preferred 'break' key is CTRL-STOP.

CTRL-J Line feed; nothing happens if this was given in the command line.

CTRL-K Home cursor (HOME).

CTRL-N This turns the printer off after being turned on by CTRL-P.

CTRL-P This turns the printer on. When on, all characters printed on the screen are also printed on the printer.

CTRL-S This suspends all character output until another key is pressed.

CTRL-U This erases the line currently being entered.

CTRL-X This erases the line currently being entered (SELECT).

The line is entered when the 'ENTER' key is pressed.

At any point whilst typing in a command line, the backspace key (marked BS or BACKSPACE on most MSX machines) can be used to delete the character immediately to the left of the cursor in the normal way.

The cursor left and right keys will move the cursor left and right along the line. Typing a character at this point will overwrite the character currently underneath the cursor.

Pressing the insert key (marked INS on most MSX machines) will toggle to 'insert mode', and the cursor will change to an underline cursor to indicate this. Instead of the characters being typed overwriting the characters under the cursor, they will instead be inserted before the cursor character, the remaining characters to the end of the line being moved one position to the right.

The delete key (marked DEL on most MSX machines) will delete the character under the cursor and move the remaining characters to the end of the line one position to the left.

The home key (marked 'HOME' on most MSX machines) will move the cursor to the start of the line.

Pressing ESC, CTRL-U or CTRL-X will clear the line to allow a new one to be entered.

The command editor also keeps a list of previous commands entered, up to a limit of 256 characters. Pressing the cursor up key will move up the list and display the previous command line entered, allowing this old command line to be edited and re-entered. Pressing the cursor down key will similarly move to the next old command line that was entered.

If a previous command line is changed, then it will be used as the new command line and added to the bottom of the list. If it was not changed, then

it will not be added to the list and the current command line will be the next one which was originally entered. This allows a whole sequence of previous commands to be entered easily.

The list of previous commands is in fact circular and moving off the top or bottom will move to the last or first command in the list respectively. The previous command can be called to be re-entered or edited from this command history list.

The features described here are in fact available to many programs that MSX-DOS can execute. In any program that does 'line inputs', each line can be edited as described above. Previous lines can be recalled for re-entering and editing, although the list of previous lines will of course include previous commands.

1.3 Notation

The syntax of the commands available from MSX-DOS are described in section 1.4 using the following notation:

- Words in upper case

  These are keywords and must be entered as shown in any mixture of upper or lower case.

- Items in lower case

  These are parameters which must be supplied to the command at this point in the command line.

- Items in square brackets (`'['` and `']'`)

  These are optional items. The brackets themselves should not be included in the command line.

- Items separatedby a vertical bar (`'|'`)

  This indicates that only one of the items is required. The vertical bar itself should not be included in the command line.

The following is a list of items which can appear on a command line:

1.3.1 `d:`

This indicates that a drive name is required (`A:`, `B:` etc.).

If `d:` is shown as optional and is not specified, then the currently logged-on drive, as indicated by the command prompt, is assumed.

1.3.2 path

This indicates that a directory path is required, the syntax of which is similar to MS-DOS. Each directory in the path is separated by a backslash '\'. A backslash at the start of the path indicates that the path starts at the root directory, otherwise the path starts at the current directory as indicated by the CHDIR command. Frequently a filename follows a path, in which case the two must be separated by a backslash.

Two consecutive dots '..' signify the immediate parent directory in the path. A single dot '.' signifies the current directory in the path and therefore usually has no value in a path specification.

On non-English MSX machines, the backslash character '\' may be replaced by some other character. In particular, on Japanese MSX machines the Yen character is used.

If a path is shown as optional and is not specified, then the current directory as indicated by the CHDIR command is assumed.

The syntax of the directory names that make up a path name follows that for filenames given below.

1.3.3 filename

This indicates that the name of a file is required, the syntax of which is similar to MS-DOS and MSX-DOS 1. An ambiguous filename is one that contains '*' and '?' characters and may match more that one file on disk, whilst one that does not contain these is an unambiguous filename.

A filename has the following syntax:

```
mainname[.suf]
```

where mainname is a sequence of up to 8 characters and suf is a sequence of up to 3 characters inclusive. Any characters beyond these fields are ignored. A '*' in the main name or suffix is equivalent to filling from that character position to the end of the field with '?'. If the suffix is given then it must be separated from the main part of the filename by a single dot '.'.

The following characters cannot be used in filenames:

- Control codes and SPACE (in the range 0 to 20h, and 7Fh and FFh)
- : ; . , = + \ < > ( ) | / " [ ]

All characters are converted to upper case where appropriate and therefore lower and upper case characters have the same meaning. Note that extended two-character Japanese characters (SHIFT-JIS code) are allowed.

If a filename is shown as optional and is not specified, then a filename of *.* is assumed.

1.3.4 filespec

This is used to identify a file or several files in the same directory on a disk. It's syntax is:

```
[d:][path][filename]
```

where at least one of the three optional items must be given. Where this is used to specify existing files, /H may be given to allow hidden files to be found.

Generally, d: if not given defaults to the currently logged on drive, path if not given defaults to the current directory of that drive and filename if not given defaults to a filename of *.*.

1.3.5 compound-filespec

This is used in many commands to specify the files or directories to which the command is applied. It's syntax is:

```
filespec [+ filespec [+ filespec ...]]
```

Thus several filespecs (see above) can be given, separated by '+' symbols, with spaces etc. allowed either side of the +. The effect of this in commands is exactly the same as if all the matched files could have been matched by a single filespec.

Where a compound-filespec is used to specify existing files, /H may be given after each filespec (see above), in which case it will take effect only for the files matched by that single filespec. If a /H is given before the compound-filespec, then it will apply over all the filespecs.

1.3.6 volname

This indicates that a volume name is required. A volume name is a sequence of up to 11 characters, which can include the characters not valid for filenames with the exception of control codes and '/', although leading spaces will be deleted.

1.3.7 device

This indicates one of the five standard MSX-DOS devices is required. These and their meaning are:

```
CON - screen/keyboard I/O
NUL - 'null' device, does nothing
AUX - auxiliary I/O (eg. RS232 serial)
LST - printer output
PRN - printer output
```

Unlike on some other systems, a colon is not required after the device name.

Device names can generally be used wherever filenames can be used. For example, the command `COPY MYFILE PRN` will read the file `MYFILE` and write it to the printer.

When using the `CON` device as an input filename, lines can be typed in and edited in the same way as command lines (see section 1.2on Editing Command Lines). To end the operation, `CTRL-Z` (`^Z`) must be typed at the start of a line. For example, a small text file called `MYFILE` can be created with the command `COPY CON MYFILE`:

```
A>COPY CON MYFILE
All work and no play makes Jack a dull boy.
Can you hear me?
^Z
A>
```

Lines of text can then be typed in, and they will be written to the file `MYFILE`. The command will then complete when a line containing a single `CTRL-Z` is entered.

If the `NUL` device is written to by the command `COPY CON NUL`, then the characters written are simply ignored. If read from, then an end-of-file condition is returned straight away (which is equivalent to typing the `CTRL-Z` in the example above).

For most commands, it is not sensible to specify a device (the `CON` device cannot be deleted using the `ERASE` command, for example). The commands that devices are likely to be used with are those that read and write data from and to files, such as `CONCAT`, `COPY` and `TYPE`.

1.3.8 number

This indicates that a number is required. This may be in the range 0 to 255 or 0 to 65535 depending on the command.

1.4 Commands

This chapter describes in detail all the commands available from the MSX-DOS CLI[1]. Each command is described using the notation described in section 1.3. Where two or more parameters are described using this notation, they must be separated by separators. Separators consist of zero or more leading spaces, a separator character, and zero or more trailing spaces. Valid separator characters are:

- `SPACE`
- `TAB`

[1] Command Line Interpreter

- ;
- ,
- =

Option letters introduced by '/' characters are an exception to this and need not be preceded by a separator.

A transient MSX-DOS or CP/M-80 program can be loaded and executed by typing the main name of the filename plus an optional extension of .COM. Batch files can similarly be executed except that the extension is .BAT. Where COM and BAT files exist in the same directory and with the same name, the COM file is found and executed in preference to the BAT file. The exact location on disk of the command can be specified by including its drive and/or path with its name.

When looking for a COM or BAT file, the specified directory of the specified drive is searched. If not found and a drive or path was given with the command, then an 'unrecognized command' error results.

If just the filename and optional extension were given, the current directory is searched first. If not found, then a list of directories is searched. This list can be specified and changed using the PATH command. If still not found then an 'unrecognized command' error again results.

No CP/M program will be able to specify directories or path names since these do not exist in CP/M, only the current directory of the appropriate drive being accessible from these programs. An APPEND environment item is available which increases the usability of these programs by allowing an alternative directory to be searched by the program as well as the current one (see section 1.7 on Environment Items).

Many commands and programs perform input or output using the 'standard input' and 'standard output'. The standard input normally refers to the keyboard, and the standard output normally refers to the screen. These can be changed, however, to refer to other devices or to disk files for the duration of the command by including the redirection symbols <, > and > > on the command line, followed by a device or file name. The standard output of one command can also be sent to the standard input of the next command by including the piping symbol | on the command line between the two commands. See section 1.5 on Redirection and Piping for more details of these facilities.

When a transient command is executed, that command may overwrite some of the memory that COMMAND2.COM was using. Thus when the command terminates, COMMAND2.COM may need to re-load itself from disk into memory from the COMMAND2.COM file that it was originally loaded from. This file is located by looking at the SHELL environment item (see section 1.7 on Environment Items), or the root directory of the boot drive if it is not found there. If it is still not found, then a prompt is issued. For example, if MSX-DOS was booted from drive A:, then the prompt will be:

```
Insert COMMAND2.COM disk in drive A:
```

```
Press any key to continue...
```

After inserting into drive `A:` a disk containing `COMMAND2.COM` in the root directory and pressing a key, `COMMAND2.COM` will be re-loaded and the system will continue as normal.

Although not a command as such, the currently logged on drive can be changed by giving the command:

```
d:
```

which causes the drive `d:` to become the current drive. This should be shown by the prompt letter.

In the command examples that follow, underlined text is an example response to a command, and the other text consists of the example command given by the user. In most examples a single space is shown as the parameter separator, although other separator characters can be used as specified above.

1.4.1 ASSIGN

- Format

  ```
  ASSIGN [d: [d:]]
  ```

- Purpose

  Sets up the logical to physical translation of drives.

- Use

  If no drives are given, then all current drive assignments are cancelled.

  If only one drive is given, then the physical drive to which this refers is printed.

  If both drives are given, then the subsequent access to the first drive (logical drive) will be done to the second drive (physical drive) by MSX-DOS.

- Examples

  ```
  ASSIGN
  ```

  Un-assigns all previous drive assignments.

  ```
  ASSIGN A: B:
  ```

  Assigns drive `A:` to drive `B:`, so that all accesses that previously would go to drive `A:` go instead to drive `B:`.

```
ASSIGN A:
A:=B:
```

Displays the drive to which A: is currently assigned, which in this case is B:.

1.4.2 ATDIR

- Format

```
ATDIR +|-H [/H] [/P] compound-filespec
```

- Purpose

  Changes the attributes of directories to make them hidden/not hidden.

- Use

  The compound-filespec specifies the directories whose attributes are to be changed.

  If +H is given, then the selected directories are marked as hidden, and will not be affected by other directory commands or shown by a DIR command unless a /H option is given with those commands. The -H option marks the selected directories as not hidden, and will not have any effect unless the /H option is given.

  Unlike files, directories cannot be made read only.

  When an error occurs, the erroneous directory name is printed followed by an error message, and the command will continue with the next directory. If many errors occur, then the /P option can be used to cause the output to pause at the end of the screen.

  The DIR /H command can be used to indicate the current attributes of directories.

- Examples

```
ATDIR +H DIR1
```

  Marks the directory called DIR1 as hidden.

```
ATDIR -H DIR1 /H
```

  Marks the hidden directory DIR1 as not hidden.

```
ATDIR +H DIR?
```

Marks all directories matching DIR? as hidden (for example DIR1, DIR2 and DIR3).

```
ATDIR +H DIR1+DIR2
```

Marks the DIR1 directory and the DIR2 directory as hidden.

1.4.3 ATTRIB

- Format

```
ATTRIB +|-R|H [/H] [/P] compound-filespec
```

- Purpose

  Changes the attributes of files to make them hidden/not hidden and read only/not read only.

- Use

  The `compound-filespec` specifies the files whose attributes are to be changed, and `/H` allows hidden files to also have their attributes changed.

  If `+H` is given, then the selected files are marked as hidden, and will not be affected by most commands or be shown by the `DIR` command unless a `/H` option is given with those commands. `-H` marks the selected files as not hidden, and will not have any effect unless the files are hidden.

  If `+R` is given, then the selected files are marked as read only. `-R` marks the selected files as not read only (read/write). Read only f iles cannot be written to or changed.

  When an error occurs, the erroneous filename is printed followed by an error message, and the command will continue with the next file. If many errors occur, then the `/P` option can be used to cause the output to pause at the end of the screen.

  The `DIR` command can be used to indicate the current attributes of files.

- Examples

```
ATTRIB +R FILE1
```

The file `FILE1` is marked as read only, and will not subsequently be modifiable or deletable.

```
ATTRIB +HB:\DIR1\*.COM
```

Marks all `.COM` files in the directory `B:DIR1` as hidden, and will not be displayed by the `DIR` command.

```
ATTRIB -R -H \DIR1/H/P
```

All the files in DIR1 are marked as not read only and not hidden. The output, if any, will pause at the bottom of the screen.

```
ATTRIB +R \DIR1 + \DIR2 + FILE1
```

All files in the directories DIR1 and DIR2 and the file FILE1 are marked as read only.

1.4.4 BASIC

- Format

```
BASIC [program]
```

- Purpose

  Transfers control to MSX disk BASIC.

- Use

  [program] is the name of a BASIC program on disk.

  Control is passed to the built-in MSX BASIC, which will load and execute the BASIC program if specified. If a RAM disk has been set up, then it can still be used from BASIC.

  The BASIC command CALL SYSTEM("command") can be used to return to MSX-DOS, and the optional command, which can be any command executable on MSX-DOS, will be executed. If the command is not given, then a batch file called REBOOT.BAT will be searched for and executed if found (see section 1.6 on Batch Files).

- Examples

```
BASIC
```

  MSX disk BASIC is entered.

```
BASIC MYPROG.BAS
```

  MSX disk BASIC is entered, and the BASIC program MYPROG.BAS loaded and run.

1.4.5 BUFFERS

- Format

```
BUFFERS [number]
```

- Purpose

  Displays or changes the number of disk buffers in the system.

- Use

  If the number is not given, then the number of disk buffers currently in the system will be displayed, otherwise the number of buffers will be changed to the specified number, the now unused memory being freed for other purposes if the new number is less than the previous. If there is not enough memory for the specified number of buffers, then as many as possible are created and no error is given.

  Increasing the number of disk buffers may speed up some applications, particularly those that perform random accesses to files. Setting the number above 10 is unlikely to improve performance much, and unnecessarily uses up memory.

  The memory area used for disk buffers is also used for environment items and for opening files. Thus keeping buffers set to the maximum possible may prevent some commands from working, particularly `SET`, `COPY` and `CONCAT`. If any of these commands give a `'not enough memory'` error then it may help to reduce the number of buffers. Reducing them below about three however will impair performance considerably.

  The default number of buffers in the system is 5, which will be adequate for most purposes.

- Examples

```
A>BUFFERS
BUFFERS=5
A>
```

  The current number of disk buffers is printed, which in this case is 5.

```
BUFFERS 10 (or BUFFERS=10)
```

  The number of buffers is increased to as many as possible up to a limit of 10.

```
BUFFERS=5 (or BUFFERS 5)
```

  The number of buffers is reduced again to 5.

1.4.6 CD

- See `CHDIR`.

1.4.7 CHDIR

- Format

```
CHDIR [d:] [path]
or
CD [d:] [path]
```

- Purpose

  Displays or changes the current directory.

- Use

  If no path is specified, then the current directory path for the current or specified drive is printed. This is the directory path from the root directory to the current directory.

  If a path is specified, then the current directory for the current or specified drive is changed to the directory specified by the path.

  Each drive has it's own current directory. This remains at the directory specified by the last `CHDIR` command for that drive (or at the root directory initially) until another `CHDIR` command is given or it cannot be found on the disk when it is accessed (because the disk has been changed, for example). It is then returned to the root directory.

  The `CD` command is an abbreviated form of the `CHDIR` command provided for convenience and MS-DOS compatibility.

  Note that the command prompt can be changed to display the current directory with the command `SET PROMPT ON` (see section 1.7 on Environment Items).

- Examples

  ```
  CHDIR \DIR1
  ```

  The current directory of the current drive is changed to `DIR1`.

  ```
  CHDIR A:DIR2
  ```

  The current directory of drive A: is changed to `DIR2`.

  ```
  A>CD
  E:\DIR1
  ```

  The current directory of the current drive is displayed, which in this case is `DIR1`.

  ```
  CHDIR A:
  A:\DIR2
  ```

  The current directory for drive A: is displayed, which is also `DIR2`.

1.4.8 CHKDSK

- Format

```
CHKDSK [d:] [/F]
```

- Purpose

  Checks the integrity of the files on the disk.

- Use

  The integrity of the data structures on disk in the specified or current drive is checked and lost disk space is checked for. When errors are found on the disk, corrective action is taken. If lost clusters are found, a prompt is issued allowing the lost disk space to be either converted into usable disk space or into files. If the latter is chosen, then files of the form `FILE0000.CHK`, `FILE0001.CHK` etc. will be created.

  If the `/F` option is not given, then `CHKDSK` will not actually write any corrections it makes to disk, but will behave as though it has. This allows `CHKDSK` to be executed to see what would be done to the disk if `/F` was given.

  Disk space can become lost (ie. lost clusters created) when some programs are aborted. This applies particularly to CP/M programs. It is recommended that `CHKDSK` is used occasionally on all disks.

  Note that this is a transient command, and must therefore be loaded off disk.

- Examples

```
CHKDSK
```

  The disk in the current drive is checked. A 'status report' will be printed. Any errors found will not be written to disk.

```
CHKDSK B:
```

  The disk in drive `B:` is checked. Any errors found will not be written to disk.

```
A>CHKDSK /F
20 lost clusters found in 1 chain
Convert lost chains to files (Y/N) ?
```

  The disk in the current drive was checked, and some lost disk space found. Since `/F` was given, the corrections will be written to disk and the lost space recovered.

1.4.9 CLS

- Format

  ```
  CLS
  ```

- Purpose

  Clears the screen.

- Use

  Simply clears the screen and homes the cursor.

- Examples

  ```
  CLS
  ```

  The screen is cleared, and another command can be typed.

1.4.10 COMMAND2

- Format

  ```
  COMMAND2 [command]
  ```

- Purpose

  Invokes the command interpreter.

- Use

  `command` is any command that can normally be typed at the prompt (such as the commands in this manual).

  `COMMAND2` is simply the name of the command interpreter on disk, and can be executed as a normal transient program. In normal use it gets loaded and executed once by `MSXDOS2.SYS` at the boot time, and this provides the ability to perform all the commands in this manual.

  Advanced users may, however, wish to invoke another command interpreter for a variety of reasons. The second `COMMAND2.COM` may, for example, be a later version and provide more facilities. If a transient program has the ability to load and execute programs, as some sophisticated programs do, then they can load the `COMMAND2.COM` program and any MSX-DOS command can then be given. When `COMMAND2.COM` exits by `EXIT` command, the original program will be returned to.

  If no command is given as a parameter, then the second `COMMAND2.COM` will simply issue the normal prompt (without executing `AUTOEXEC.BAT` or `REBOOT.BAT`) and wait for commands in the normal way. It will terminate and exit back to the original command interpreter or program when the `EXIT` command is given (see the `EXIT` command on page 36). If an

error code is given to this EXIT command, then the original command interpreter or program will receive it and, in the case of COMMAND2.COM and MSXDOS2.SYS, print an appropriate error message (see section 1.8 on Errors).

If a command is given as a parameter to COMMAND2.COM however, then it will be executed as though it had been typed in the normal way. The command may be an internal command or an external COM or BAT command. After executing the command, COMMAND2.COM will immediately exit back to the original command interpreter or program.

In this way, invoking a second COMMAND2.COM from the normal command interpreter with a batch file name as a command can be used to 'nest' batch files (see section 1.6 on Batch Files), instead of 'chaining'.

When COMMAND2.COM is executed, it saves the whole environment, and then restores it again when it exits. It only sets up the default environment items however if they are not already defined. Thus the second COMMAND2.COM inherits the environment of the first. Any changes made whilst the second COMMAND2.COM is executing will only last as long it does, and will be lost when it exits.

Each incarnation of COMMAND2.COM uses up some memory which is freed again when it exits. This depends partly on the number of environment items, and is typically about 1.5K.

When COMMAND2.COM executes a transient program, it may have to re-load itself off disk since the program is allowed to use the memory occupied by COMMAND2.COM. In this case, it uses the SHELL environment item to locate the file that it must use to load itself (see section 1.7 on Environment Items). When first loaded from the COMMAND2.COM file on disk, SHELL is set to refer to that file.

- Examples

```
COMMAND2
A>
```

Another copy of COMMAND2 is loaded, and prints it's normal prompt. EXIT will exit back to the original prompt.

```
COMMAND2 FILE.BAT
```

Normally in a batch file. The file FILE.BAT is executed, and when it ends the current batch file will be resumed with the command after this one.

1.4.11 CONCAT

- Format

```
CONCAT [/H] [/P] [/B] [/V] compound-filespec filespec
```

- Purpose

  Concatenates (joins together) files.

- Use

  The `compound-filespec` specifies the files that are to be joined together, and `/H` allows hidden files to be joined.

  The second parameter is a `filespec` that must be unambiguous and is created before the source files are read. Each source file is then read, joined onto the end of the previous one and written out to the destination.

  As each source file is read, its filename is printed. If for some reason the file cannot be read (eg. it is the file that has been created as the destination) then the filename is followed by an error message and the `CONCAT` operation continues with the next source file. If many files are being concatenated, then `/P` will cause the output to pause at the end of the screen until a key is pressed.

  Normally, the concatenation is performed on ASCII files. Source files are read up to the first end-of-file character (`CTRL-Z`) and a single end-of-file character is appended to the destination after all data has been written out. If, however, `/B` (binary mode) is given, then no interpretation is given to the data read and no additional data is added.

  It is also possible to give the `/B` to the destination or to any of the `filespecs` in the `compound-filespec`, and it will then refer only to those files. `/A` may also be given to reverse the effect of `/B`.

  The `/V` option can be given to turn write verification on for the duration of the `CONCAT` command (see the `VERIFY` command). This will ensure that data is written correctly to disks if the device driver being used has the feature, but will slow the operation down for the verification.

  If `CONCAT` gives a `'Not enough memory'` error then probably reducing the number of buffers (see the `BUFFERS` command) or removing some environment items (see section 1.7 on Environment Items) will free up sufficient memory.

- Examples

  ```
  CONCAT *.DOC ALL.PRN
  ```

  A new file called `ALL.PRN` is created, and all files matching `*.DOC` (for example `FILE1.DOC`, `FILE2.DOC` and `FILE3.DOC`) are joined together and written to the new file in the order that they are found on disk. Any existing file called `ALL.PRN` will be overwritten.

```
A>CONCAT /H /P *.DOC ALL.DOC
FILE1.DOC
FILE2.DOC
FILE3.DOC
ALL.DOC -- Destination file cannot be concatenated
```

A new file called `ALL.DOC` is created, and all files matching `*.DOC` are joined together and written to the new file in the order that they are found on disk. Since the destination file `ALL.DOC` also matched the source filename `*.DOC`, the message is printed and it is not included in the concatenation. Since `/H` was given, hidden files are also concatenated and, since `/P` was given, the key input is waited after each screenful output if the number of the lines of the file list is larger than that of the screen.

```
CONCAT /B FILE2.DOC + FILE3.DOC + FILE1.DOC ALL.DOC
```

A new file `ALL.DOC` is created, and the files `FILE2.DOC`, `FILE3.DOC` and `FILE1.DOC` and joined together in that order and written to the new file. They are joined together in binary mode.

1.4.12 COPY

- Format

```
COPY [/A] [/H] [/T] [/V] [/P] [/B] source dest
```

- Purpose

  Copies data from files or devices to other files or devices.

- Use

  The definition of the `source` is:

```
compound-filespec | device
```

  The `compound-filespec` specifies the files that are to be copied. It may be the device specification. If `/H` is given then hidden files may be copied.

  The definition of the `dest` is:

```
[d:] [path] [filename] | device
```

  Where `d:` and path default to the current drive and directory respectively. If any part of the filename is ambiguous then the appropriate character from the source filename is substituted, thus allowing the files to be renamed in the process. If the filename is not given, then the entire source filename is used. If the dest is an unambiguous directory, then the files are copied into that directory with a filename of `*.*`.

COPY will read as many source files as possible into memory before writing any out. When it can read no more into memory (eg. when it has used all available memory) it will write out each file in the order that it read them. When it creates each destination file, it prints the source filename. Then if it is unable to create the destination file, an error message is printed and the copy operation continues with the next file. /P can be given to make the output pause at the end of the screen.

Many reasons exist for COPY to be unable to create the destination, such as a read-only file already existing with the same name. Sometimes COPY will refuse to create the destination because the user may have made a mistake. For example, a file cannot be copied onto itself, or several files cannot be copied onto one file. A 'Cannot create destination' error may be given if the destination of one file would delete a previous source file or a file already being used for something else (eg. the currently executing batch file). A 'Cannot overwrite previous destination file' error results if an attempt is made to copy many files to one file. This usually means that the intended destination was a directory, but that the name has been misspelled.

If /A is specified, then an ASCII copy is performed. This means that source files will only be read as far as the first end-of-file (EOF) character (CTRL-Z) and then each destination will have a single end-of-file character appended to it.

It is also possible to give a /A to the destination or to any of the filespecs in the compound-filespec separately, in which case it applies only to that source or dest specification.

The /B option can be given to copy in binary mode, that is, the file being read will be copied as it is and no data will be added.

The /V option can be given to turn write verification on for the duration of the COPY command (see the VERIFY command). This will ensure that data is written correctly to disks if the device driver being used has the feature, but will slow the operation down.

Normally, the destination files are given the same date and time as the source files. However, the /T option can be given to cause the destination files to have the current date and time. The destination files will not be hidden or read-only, regardless of the attributes of the source files. The ATTRIB command can be used to change these.

If COPY gives a 'Not enough memory' error then probably reducing the number of buffers (see the BUFFERS command) or removing some environment items (see section 1.7 on Environment Items) will free up sufficient work area.

Note that the COPY command is simpler than that in MS-DOS and MSX-DOS 1 because it cannot concatenate (join together) files. To do this, a CONCAT command is available (see the CONCAT command).

- Examples

```
COPY FILE1 B:
```

The file FILE1 is copied from the current directory of the current drive to the current directory of drive B:.

```
COPY /H MSXDOS2.SYS + COMMAND2.COM B:
```

The two hidden files MSXDOS2.SYS and COMMAND2.COM are copied to drive B:, thus making it a booting disk.

```
COPY A:\DIR1 B:\DIR1 /V
```

All files in the directory DIR1 from the root of drive A: are copied to a similar directory on drive B: with verify on to ensure that the files were written correctly.

```
COPY B:
```

All files in the current directory of drive B: are copied to the current directory of the current drive.

```
COPY /A AUX CON
```

Characters are read from the device AUX (which may be used for RS232 serial for example) to the device CON, which is the screen. The copy is done as far as the first end-of-file character. If /A was not given, then there may have been no way of stopping the COPY operation without pressing the CTRL-STOP key.

```
COPY A:*.DOC B:/T
```

All files matching *.DOC (for example FILE1.DOC, FILE2.DOC and FILE3.DOC) are copied to the current directory of drive B: and are given the current date and time instead of the dates and times of the *.DOC files.

```
A>COPY *.BAT
AUTOEXEC.BAT -- File cannot be copied onto itself
REBOOT.BAT -- File cannot be copied onto itself
```

This command told COPY to copy all files matching *.BAT (in this case AUTOEXEC.BAT and REBOOT.BAT) from the current directory of the current drive to the same place, and COPY printed the messages to warn of this. No data in this case was actually copied.

```
A>COPY *.BAT DIR2
AUTOEXEC.BAT
REBOOT.BAT -- Cannot overwrite previous destination file
```

This command told COPY to copy all files matching *.BAT (in this case AUTOEXEC.BAT and REBOOT.BAT) to a directory called DIR2. DIR2, however, did not exist so AUTOEXEC.BAT was copied to a file called DIR2, and then an attempt was made to copy REBOOT.BAT also to a file called DIR2. The message was printed as a warning that a mistake was probably made (in this case DIR2 not existing). REBOOT.BAT was not actually copied anywhere.

1.4.13 DATE

- Format

  ```
  DATE [date]
  ```

- Purpose

  Displays or sets the current date.

- Use

  If the date is given after the command, then the date is set to this value (for the format see below). If the date is not given after the command, then the current day and date is printed and the new date is prompted for and input. If no input is given (ie. if the 'ENTER' key alone is pressed) then the current date is not altered. Otherwise the input is assumed to be a new date, and is interpreted as described below. If the date is invalid then an error message is displayed and the new date again prompted for and input.

  The date is expected to consist of up to three numbers, separated by one of the following characters:

  ```
  SPACE TAB , - . / :
  ```

  with spaces allowed either side of the character. Any missing numbers will default to the current setting. The year may either be a full century and year, or may be just the year in which case the century defaults to 19 if the year is greater than 80 or 20 otherwise. The date and the year specifications may be substituted by '-' to be omitted.

  The format in which the date is printed and input is flexible and can be changed. An environment item called DATE is set up by default to a format that is appropriate for the country of origin of the MSX machine (see section 1.7 on Environment Items). For example, on Japanese machines the default setting is YY-MM-DD. The command SET DATE DD-MM-YY will change the date format to the European format. The format also affects the dates printed by the DIR command.

  If the DATE environment item is defined, then it will be printed by the DATE command to indicate the format in which the date is required to be input.

- Examples

```
DATE 86-6-18
```

The current date is set to the 18th June 1986.

```
A>DATE
Current date is Wed 1986-06-18
Enter new date (yy-mm-dd): - -19
```

No parameter was given, so the current date of 18th June 1986 was printed and a new date prompted for. In the reply to the prompt, the date was updated to the next day by only specifying the 19th. Since the year and month were not given, they remained the same.

```
SET DATE = DD/MM/YY
```

The date format has been changed to the European format.

```
A>DATE
Current date is Thu 19-06-1986
Enter new date (DD/MM/YY):
A>
```

No parameter was given, so the current date of 19th June 1986 was printed in the European format, and the prompt printed. The reply is expected in the European format.

Formats are:

ISO	`YY/MM/DD`
American	`MM/DD/YY`
European	`DD/MM/YY`

1.4.14 DEL

- See `ERASE`.

1.4.15 DIR

- Format

```
DIR [/H] [/W] [/P] [compound-filespec]
```

- Purpose

Displays the names of files on disk.

- Use

  The `compound-filespec` specifies which files are to be listed. If the `/H` option is given, then hidden files will also be listed.

  In the `DIR` command, unlike all other commands, it is permissible to not give the main filename or the filename extension, and both will default to `'*'`. Thus a filename of `'FRED'` is equivalent to `'FRED.*'` and a filename of `'.COM'` is equivalent to `'*.COM'`. Note that if the `'.'` at the end of a main filename is given, then the extension is also assumed to have been given, so that the filename `'FRED.'` is not equivalent to `'FRED.*'`, unlike the example above.

  There are two formats of the listing. If the `/W` option is given, then a *wide* listing is printed, with several filenames output per line. Sub-directory names, file attributes, and the date and time each file was created are not displayed.

  If the `/W` option is not given, then the filenames are printed with one filename per line, together with the attributes, the file size and the date and time at which the file was last modified. The attributes are printed as an `'r'` if the file is read-only and an `'h'` if the file is hidden (and `/H` is given). If the time of a file is zero (ie. the file does not have an associated time) then the time field will not be printed. If the date of a file is zero, then neither the date nor the time fields will be printed. The formats in which the dates and times are printed can be changed (see the `DATE` and `TIME` commands).

  The non-`/W` display is designed to fit within a 40 column screen, but if fewer columns are available then some fields of the listing will not be shown so that the display will always fit on one line. The number of files per line that are printed when `/W` is specified is also adjusted according to the screen width. If the width of the display is less than 13 characters however, then in both cases the filenames will wrap to the next line.

  At the top of the list of files, the volume name of the disk and the name of the directory being listed is displayed. At the bottom, the number of files listed, the total number of bytes in the files and the amount of remaining disk space is printed.

  When the directory of a sub-directory is printed, the first two items listed will always be two special sub-directories called `'.'` and `'..'`. These are automatically created when a new directory is created, and it is these that allow `'.'` and `'..'` to be given in path names to signify the current and parent directories respectively (see section 1.3 on Notation for a description of paths).

  When printing a number of bytes, the number is truncated and printed as the number of kilobytes if 1K or greater.

  If the `/P` option is given, then the output will pause at the bottom of the screen until a key is pressed.

- Examples

```
DIR
```

All filenames and directory names in the current directory of the current drive will be printed. This might be as follows:

```
 Volume in drive A: is MSX-DOS 2
 Directory of A:\

MSXDOS2  SYS r   4096 86-06-19 2:45p
COMMAND2 COM r  10496 86-06-19 2:46p
UTILS          <dir>  86-06-19 2:50p
HELP           <dir>  86-06-19 2:50p
 14K in 2 files  222K free
```

The disk thus contains the two MSX-DOS system files `MSXDOS2.SYS` and `COMMAND2.COM`, which are read only, and two directories called `UTILS` and `HELP`.

```
DIR B:\HELP /W
```

A 'wide' directory format has been requested of the `HELP` directory on drive `B:`. This might be as follows:

```
 Volume in drive B: is MSX-DOS 2
 Directory of B:\HELP

BUFFERS .HLP    ATTRIB .HLP    ASSIGN .HLP
ATDIR   .HLP    CHDIR  .HLP    CD     .HLP
SYNTAX  .HLP    ENV    .HLP    BATCH  .HLP
EDITING .HLP
 25K in 10 files 222K free
```

```
DIR UTILS + HELP /P
```

This will list all the files in the `UTILS` directory and all the files in the `HELP` directory, and will pause at the end of every screen full.

```
DIR .COM
```

No main filename was given, and so defaults to `*`. Thus this command is equivalent to the command `DIR *.COM`.

```
DIR COMMAND2
```

No extension was given, so this defaults to `.*`. Thus this command is equivalent to the command `DIR COMMAND2.*`.

1.4.16 DISKCOPY

- Format

```
DISKCOPY [d: [d:]] [/X]
```

- Purpose

  Copies one disk to another.

- Use

  The first drive is the source drive and the second the destination, which defaults to the current drive. If no drives are given, then DISKCOPY will prompt for both the source and the destination.

  Before DISKCOPY is used, the destination disk must be formatted with the same format as the source disk, and an error will be given if this is not the case.

  If /X is given, then various messages printed during the disk copy operation will be suppressed.

  Note that this is a transient command, and must therefore be loaded from disk.

- Examples

```
A>DISKCOPY A: B:
Insert source disk in drive A:
Insert target disk in drive B:
Press any key to continue...
```

  The command was given to copy the disk in drive A: to the disk in drive B:, thus destroying all existing data on the disk in drive B: The prompt is printed first.

```
DISKCOPY B:
```

  The disk in drive B: is copied to the disk in the current drive.

```
A>DISKCOPY
Enter source drive:
Enter target drive:
```

  The DISKCOPY command was given with no parameters, so the source and destination disks were prompted for. The reply to the prompts consists of just a single drive letter.

1.4.17 ECHO

- Format

```
ECHO [text]
```

- Purpose

  Prints text.

- Use

  The text is simply displayed on the screen. If no text is given, then just a blank line is output.

  This command should not be confused with the *echo* state in batch files, which is controlled by an environment item called `ECHO` (see section 1.7 on Environment Items).

- Examples

```
A>ECHO AUTOEXEC batch file executed
AUTOEXEC batch file executed
```

  The specified text (`'AUTOEXEC batch file executed'`) was printed on the screen.

```
ECHO
```

  No parameters were given, so just a blank line was printed.

1.4.18 ERA

- See `ERASE`.

1.4.19 ERASE

- Format

```
ERASE [/H] [/P] compound-filespec
```

  or

```
DEL [/H] [/P] compound-filespec
```

  or

```
ERA [/H] [/P] compound-filespec
```

- Purpose

  Deletes one or more files.

- Use

  The `compound-filespec` specifies which files are to be deleted. The `/H` option allows hidden files to also be deleted.

  During the delete operation, if a file cannot be deleted for some reason (eg. it is set to `'read only'`) then the offending filename is printed along with an error message, and the delete operation continues with the next file. If many such errors occur, then the `/P` option will cause the output to pause at the end of the screen.

  If the filename is `*.*`, then the prompt:

  ```
  Erase all files (Y/N) ?
  ```

  is printed, and a reply is waited for. If the reply is anything other than `'Y'` or `'y'`, then the file deletion does not take place. This is a safety feature designed to prevent accidental loss of all files in a directory.

  If files are deleted unintentionally on a disk that was formatted under MSX-DOS 2, then the `UNDEL` command may be used immediately afterwards to restore them again.

- Examples

  ```
  ERASE FILE1.BAK
  ```

  The file `FILE1.BAK` is deleted from the current directory of the current drive.

  ```
  DEL *.COM/H
  ```

  All files matching `*.COM`, both hidden and not hidden, are deleted.

  ```
  DEL B:\UTIL\*.COM + B:\UTIL\*.BAT
  ```

  All files matching *.COM or *.BAT are deleted from the directory called UTIL on drive B:.

  ```
  A>DEL B:\UTIL
  Erase all files (Y/N) ?
  ```

  All files in the directory called `UTIL` on drive `B:` are deleted. Since so many files are being deleted, a prompt is printed first to prevent a catastrophe.

  ```
  A>DEL *.BAT
  AUTOEXEC.BAT -- Read only file
  REBOOT.BAT -- Read only file
  ```

  All files matching `*.BAT` are deleted except for `AUTOEXEC.BAT` and `REBOOT.BAT` which have been marked as read only.

1.4.20 EXIT

- Format

```
EXIT [number]
```

- Purpose

  Exits COMMAND2.COM to the invoking program.

- Use

  The number is an error code and defaults to 0, which in MSX-DOS indicates no error (see section 1.8 on Errors).

  EXIT exits the command interpreter (COMMAND2.COM) and returns the error code to the program that originally loaded and executed it (see the COMMAND2 command). This may be another COMMAND2.COM, another program or, normally, MSXDOS2.SYS. In the latter case, an appropriate error message will be printed and COMMAND2.COM simply reloaded and executed.

  COMMAND2.COM when loaded saves the current environment (see section 1.7 on Environment Items), and EXIT restores it. Thus, when EXIT exits back to MSXDOS2.SYS (that is, EXIT is executed at the primary level), the environment will be cleared. COMMAND2.COM will then be reloaded and will set up the default environment again, providing a method of resetting the environment to its default values.

- Examples

```
EXIT
```

  The command interpreter is exited. What happens next depends on what loaded it.

```
A>EXIT 40
*** User error 40
```

  The command interpreter is exited with an error code of 40. Since this does not correspond to an error that is known to the system, the error message is printed by whatever loaded the command interpreter in the first place (see section 1.8 on Errors).

1.4.21 FIXDISK

- Format

```
FIXDISK [d:] [/S]
```

- Purpose

  Updates a disk to the full MSX-DOS 2 format.

- Use

  `d:` specifies the drive on which `FIXDISK` is to operate. If `d:` is not specified, the current drive will be assumed. `/S` option causes the disk to be updated to full MSX-DOS 2 disk.

  This command is mainly used to update MSX-DOS 1 disks to full MSX-DOS 2 compatibility, but may also be useful for updating other disks of a similar format or reparing incorrect boot sector.

  Although the disk format used by MSX-DOS 1 and MSX-DOS 2 is standardized, MSX-DOS 1 does not use the information stored on certain parts of the disk (the boot sector) and so this information is not necessarily correct on MSX-DOS 1 disks. This can cause problems when MSX-DOS 2 is used with these disks. Additionally, the MSX-DOS 2 `UNDEL` command will only work with disks that were formatted using MSX-DOS 2 (ie. disks that have a '*volume id*' in the boot sector) and so will not work with MSX-DOS 1 disks or disks formatted on other systems.

  The `FIXDISK` command will update a disk so that it is fully MSX-DOS 2 compatible, and its use will allow full use of MSX-DOS 2 disk features. If `/S` option is specified, the boot program will be updated for MSX-DOS 2 so that the features of MSX-DOS 2 disk can fully work. When a disk has been updated in this way, however, it may no longer be fully compatible with the original system. For example, if `/S` option is specified against the disk of the application which uses non-standard boot program, such as some games, the application will no longer be booted, although it will still be able to start MSX-DOS 1 or MSX-DOS 2.

  To help prevent accidental updates of boot disks from other systems, a prompt is issued before updating a disk.

- Examples

```
FIXDISK B: /S
Disk in drive B: will only be able to boot MSX-DOS 2
Press any key to continue...
```

  Drive `B:` will be updated to be fully MSX-DOS 2 compatible. Since the disk may have been a boot disk from another system, a prompt is issued before the disk is actually updated.

1.4.22 FORMAT

- Format

```
FORMAT [d:]
```

- Purpose

  Formats (initializes) a disk.

- Use

  The specified or current drive is formatted, and all data on the disk will be destroyed.

  After giving a `FORMAT` command, an option may be prompted for, allowing the required format of the disk (such as 1DD or 2DD) to be selected. The exact nature of these prompts depends on the manufacturer of the MSX machine, so obey the descriptions of the manual of the machine when you want to format the disk.

  After formatting, there will be no files or directories on the disk, and the maximum amount of disk space will be free. The disk will not have a volume name, but can be given one with the `VOL` command. To turn the disk into a boot disk so that MSX-DOS can be started up from it, the files `MSXDOS2.SYS` and `COMMAND2.COM` must be copied onto it with the `COPY` command.

- Examples

```
A>FORMAT B:
1 - Single sided
2 - Double sided
? 2
All data on drive B: will be destroyed
Press any key to continue...
```

  The command was given to format the disk in drive `B:`. In this case, the options available were to select either double sided or single sided, and double sided was selected. The standard warning prompt was then printed.

```
FORMAT
```

  This will format the current drive after the prompts given above.

1.4.23 HELP

- Format

```
HELP [subject]
```

- Purpose

  Provides on-line help for an MSX-DOS feature.

- Use

  If no parameter is given, then a list of standard subjects on which help is available is printed. This includes all the commands and the major system features.

If a subject is specified, then help text on this subject is printed on the screen from a '*help file*'. This will pause at the end of every screen until a key is pressed.

Help files have a filename of:

```
subject.HLP
```

and are located by default on the standard MSX-DOS boot disk in a directory called `HELP`.

An environment item called `HELP` is set up initially to refer to the `HELP` directory (see section 1.7 on Environment Items). This can be changed with the `SET` command to refer to any other directory or disk if required.

Any `HELP` subject can be added by the user for his own use simply by adding the appropriate `.HLP` file in the `HELP` directory. The help file will be displayed very much like the `TYPE` command would display it.

- Examples

```
HELP
```

A general help screen is printed. This lists all the subjects on which help is available, including the standard commands and main features of MSX-DOS. The user-supplied subjects are not available here.

```
HELP XCOPY
```

Help information on the `XCOPY` command is printed. This includes a description of how to use the command and what options are available.

```
A>HELP ME
*** File for HELP not found
```

This command caused `HELP` to look for a file called `ME.HLP` from which to get the help text, but did not find it so printed the error message. The files containing the help text are normally found in a directory called `\HELP` on the drive from which MSX-DOS was booted, and any other help files may be added if required. If `ME.HLP` was added then `HELP ME` would print it on the screen.

1.4.24 MD

- See `MKDIR`.

1.4.25 MKDIR

- Format

```
MKDIR [d:] path
```

or

```
MD [d:] path
```

- Purpose

Creates a new sub-directory.

- Use

The last item in the path is the name of the new sub-directory which is to be created on the current or specified drive. Thus if this is the only item in the path, the new directory is created in the current directory. If the new directory is to be hidden, then it must be separately made hidden with the ATDIR command.

When a new directory is created, it is empty except for two special sub-directories called '.' and '..'. These are automatically created in the directory and it is these that allow '.' and '..' to be given in path names to signify the current and parent directories respectively (see section 1.3 on Notation for a description of paths).

The MD command is an abbreviated form of the MKDIR command provided for convenience and MS-DOS compatibility.

- Examples

```
MKDIR UTIL
```

A directory called UTIL is created in the current directory of the current drive.

```
MKDIR A:\UTIL\RAM
```

A directory called RAM is created in the UTIL directory in the root directory of drive A:.

1.4.26 MODE

- Format

```
MODE number
```

- Purpose

Changes the number of characters/line on the screen.

- Use

  The number must be in the range 1 to 80 inclusive, and the number of characters per line on the screen will be set to this. The screen will be cleared and the cursor moved to the top left corner in the process.

- Examples

```
MODE 80 (or MODE=80)
```

  The screen is set to 80 column mode and is cleared in the process.

```
MODE 25 (or MODE=25)
```

  The screen is set to 25 columns.

1.4.27 MOVE

- Format

```
MOVE [/H] [/P] compound-filespec [path]
```

- Purpose

  Moves files from one place to another on a disk.

- Use

  The `compound-filespec` specifies which files are to be moved, and `/H` allows hidden files to be included in the move.

  The `path` specifies the directory to which the files are to be moved, the current directory being used if this is not given. The path must exist on each drive referenced by the `filespecs` in the `compound-filespec`.

  If a particular file cannot be moved into the specified or current directory (eg. if a file of the same name already exists) then the offending filename is printed along with an error message, and the move operation continues with the next file. If many errors occur, then the `/P` option will cause the output to pause at the bottom of the screen.

- Examples

```
MOVE FILE1 \
```

  The file `FILE1` is moved from the current directory of the current drive to the root directory of the current drive.

```
MOVE /H /P E:*.COM \
COMMAND2.COM -- File exists
```

All files matching `*.COM`, both hidden and not hidden, in the current directory of drive `E:` are moved to the root directory of that drive. The file `COMMAND2.COM` already existed in the root directory, so the error was printed. Neither of the `COMMAND2.COM` files were moved or altered. If many such errors had occurred then a prompt would have been printed after a screen full.

```
MOVE \UTIL\*.COM + \UTIL\*.BAT
```

All files matching `*.COM` or `*.BAT` in a directory called `UTIL` on the current drive are moved to the current directory of that drive.

1.4.28 MVDIR

- Format

```
MVDIR [/H] [/P] compound-filespec [path]
```

- Purpose

  Moves directories from one place to another on a disk.

- Use

  The `compound-filespec` specifies which directories are to be moved, and `/H` allows hidden directories to be included in the move.

  The second parameter specifies the directory into which the directories are to be moved, the current directory being used if this is not given. The path must exist on each drive referenced by the `filespecs` in the `compound-filepecs`.

  If a particular directory cannot be moved into the specified or current directory (eg. if a directory of the same name already exists) then the offending directory name is printed along with an error message, and the move operation continues with the next directory. If many errors occur, then the `/P` option will cause the output to pause at the bottom of the screen.

  Note that it is not possible to move a directory into one of its own descendant directories, as this would produce an invalid sub-directory tree. An error is given if this is attempted.

- Examples

```
MVDIR COM UTIL
```

  A directory called `COM` and all descendant directories and files are moved into a directory called `UTIL`, both directories being in the current directory of the current drive.

```
MVDIR \COM + \BAT \UTIL
```

A directory called COM and a directory called BAT, and both their contents, are moved into a directory called UTIL.

```
MVDIR E:DIR?/H/P ALL
DIR2 -- Duplicate filename
```

All directories in drive E matching DIR? (eg. DIR1, DIR2 and DIR3), which may be hidden, and the contents of the directories, are moved into a directory called ALL. A directory called DIR2 already existed in ALL so the error was printed. Neither of the DIR2 directories were affected at all.

1.4.29 PATH

- Format

```
PATH [ [+|-] [d:]path [ [d:]path [ [d:]path ...]] ]
```

- Purpose

  Displays or sets the COM and BAT command search path

- Use

  If no parameters are specified, then the search path currently set will be displayed, separated by semi-colons (';').

  If + or - is not given, then the search path will be set to the list of path names given and any existing search path will be deleted.

  If - is given before the list of paths, then each path in the list will be deleted from the currently set search path, and an error will be given if any of the given paths do not already exist.

  If + is given before the list of paths, then each path specified will first be deleted from the currently set search path if it exists, and will then be added onto the end. This allows the order of the paths in the search path to be changed and allows new paths to be appended to the end of the current search path. The + syntax can also be used to set a search path longer than can be given in one command, the maximum length of the search path being 255 characters and the maximum length of a command 127 characters.

  When searching for a COM or BAT file, the paths in the current search path will be used in order from left to right. It is recommended that the paths in the search path are specified as full paths starting at the root directory and with the drive specified. If this is not the case, then the meaning of the search path could change when the current drive or directory is changed.

  The search path is stored as an environment item (see section 1.7 on Environment Items), and so can also be accessed with the SET command.

- Examples

```
PATH E:\COM E:\BAT
```

When a `COM` or `BAT` command is next searched for, the directories searched will be the current directory of the current drive, the `COM` directory in the root directory of drive `E:` and the `BAT` directory in the root directory of drive `E:`, in that order.

```
A>PATH
; E:\COM; E:\BAT
```

No parameters were given so the current search path was printed.

```
PATH +A:\COM; A:\BAT
```

The directories `A:\COM` and `A:\BAT` are added to the end of the search path.

```
A>PATH
; E:\COM; E:\BAT; A:\COM; A:\BAT
```

The new search path is printed.

```
PATH -E:\COM, E:\BAT
```

The directories `E:\COM` and `E:\BAT` are deleted from the current search path.

```
A>PATH
; A:\COM; A:\BAT
```

The new search path is again printed.

1.4.30 PAUSE

- Format

```
PAUSE [comment]
```

- Purpose

Prompts and waits for a key press in a batch file.

- Use

The comment consists of an arbitrary sequence of characters.

The comment, if given, is printed followed by the prompt `'Press any key to continue... '` on the next line. The system will then wait for a key

to be pressed and will echo the key pressed if it is a printable character. If no comment is given as a parameter, then just the prompt will be printed.

The main use of this command is to issue prompts from within a batch file.

- Examples

```
PAUSE
Press any key to continue...
```

No comment was given, so just the prompt was printed.

```
PAUSE Insert document disk in drive B:
Insert document disk in drive B:
Press any key to continue...
```

The comment given was `'Insert document disk in drive B:'` so this was printed followed by the prompt.

1.4.31 RAMDISK

- Format

```
RAMDISK [number[K]] [/D]
```

- Purpose

Displays or sets the RAM disk size.

- Use

If no parameters are given, then the current `RAMDISK` size is displayed as the number of kilobytes.

The number, if given, specifies the maximum size for the new RAM disk, and is specified in kilobytes. The range is 0 to 4064. If the number is 0 or only `/D` is specified, the RAM disk will be deleted. This number will be rounded up to the nearest multiple of 16K since the RAM disk is always a multiple of 16K. A RAM disk smaller then the specified maximum size may be created if there is not enough free memory for the full size, although a `'not enough memory'` error will be given if there is no memory at all available for the RAM disk. Note that the number specified is the maximum amount of RAM to use for the RAM disk, which is not the same as the maximum amount of free space available on the newly-created RAM disk since the system needs to use some for FAT or directories.

On MSX machines with 128K RAM, the maximum amount of RAM disk is 32K.

If a RAM disk already exists before a new one is created, then a `'Destroy all data on RAM disk (Y/N)?'` prompt is printed to avoid accidental

loss of data. /D can be given which will automatically delete any existing RAM disk first, thus suppressing the prompt.

Having created a RAM disk, it can be referred to as drive H:.

The RAMDISK command is normally only used in an AUTOEXEC.BAT batch file, with a large number specified so that as large a RAMDISK as possible is created. It is not advisable to keep any data on a RAM disk except for a short length of time that is not also kept on a floppy disk, since it will be lost if, for example, the power to the computer fails.

- Examples

```
A>RAMDISK
RAMDISK=160K
```

No parameters were given, so the current size is printed, in this case 160K.

```
A>RAMDISK
*** RAM disk does not exist
```

No parameters were given but no RAM disk has been created, so the error is given.

```
A>RAMDISK = 300
Destroy all data on RAM disk (Y/N)? y
```

A RAM disk already existed, so the prompt was printed. In this case, the reply was 'y' so the current RAM disk was deleted and the new one set up with a maximum size of 300K.

1.4.32 RD

- See RMDIR.

1.4.33 REM

- Format

```
REM [comment]
```

- Purpose

Introduces a comment in a batch file.

- Use

The comment is simply ignored, and the next command executed. The comment consists of a sequence of any characters up to the maximum length of a command line (127 characters).

- Examples

```
REM This is my AUTOEXEC batch file
```

This command, either in a batch file or typed in, does nothing at all with it's parameters.

1.4.34 REN

- See RENAME.

1.4.35 RENAME

- Format

```
RENAME [/H] [/P] compound-filespec filename
```

or

```
REN [/H] [/P] compound-filespec filename
```

- Purpose

Renames one or more files.

- Use

The compound-filespec specifies the files that are to be renamed, and /H allows hidden files to be included in the rename operation.

The second filename specifies the new name for the files. A '?' in the new name indicates that the corresponding character from the filename being renamed will be used, thus allowing an ambiguous rename. Thus '*' in the second filename, which is just equivalent to a series of '?'s, indicates that the whole of the filename or extension will remain unchanged.

If for some reason a particular file cannot be renamed (eg. if a file or directory with the new name already exists or they are read-only) then the offending filename will be printed along with an error message and the rename operation will continue with the next file. If many errors occur, then /P will cause the output to pause at the end of the screen.

- Examples

```
RENAME FILE1 FILE2
```

The file FILE1 in the current directory of the current drive is renamed to FILE2.

```
REN B:\DIR1\*.DOC/H/P *.OLD
FILE2.DOC -- Duplicate filename
```

All files matching `*.DOC` in the directory called `DIR1` in the root directory of drive `B:`, including hidden files, are renamed with the same main name but with an extension of `.OLD`. The file `FILE2.DOC` could not be renamed because there was already a file called `FILE2.OLD` in the directory, so the error was printed. Neither `FILE2.DOC` nor `FILE2.OLD` was altered at all. If many such errors had been printed, then a prompt would have been printed at the bottom of every screen full, since `/P` was given.

```
REN DOC + FILE1 *.OLD
```

All files in the directory called `DOC` and the file `FILE1`, both in the current directory of the current drive, and renamed with an extension of `.OLD`.

1.4.36 RMDIR

- Format

  ```
  RMDIR [/H] [/P] compound-filespec
  ```

  or

  ```
  RD [/H] [/P] compound-filespec
  ```

- Purpose

  Removes one or more sub-directories.

- Use

  The `compound-filespec` specifies which directories are to be deleted, and `/H` allows hidden directories to be included in the delete operation.

  In order to delete a directory, it must contain no other files or other directories except for the special `'.'` and `'..'` directories which are always contained in a directory. These are put in a new directory when it is created and cannot be removed. It is these that allow `'.'` and `'..'` to be used in path names to specify the current and parent directories respectively (see section 1.3 on Notation for a description of paths).

  If a directory cannot be deleted for some reason (eg. it is not empty) then the name of the offending directory is printed along with an error message, and the delete operation continues with the next directory. If many errors occur then `/P` will cause the output to pause at the end of the screen.

- Examples

  ```
  RMDIR DIR1
  ```

  The directory called `DIR1` in the current directory of the current drive is removed.

```
RD B:\COM + B:\BAT
```

The directories `COM` and `BAT` are removed from the root directory of drive `B:`.

```
RD \*.*
UTIL -- Directory not empty
```

An attempt was made to remove all directories from the root directory of the current drive, but a directory called `UTIL` was not empty and so the error was printed. `UTIL` and its contents are not affected at all.

1.4.37 RNDIR

- Format

  ```
  RNDIR [/H] [/P] compound-filespec filename
  ```

- Purpose

  Renames one or more sub-directories.

- Use

  The `compound-filespec` specifies the directories that are to be renamed, and `/H` allows hidden directories to be included in the rename operation. The contents of the directories remain unchanged.

  The second filename specifies the new name for the directories. A `'?'` in the new name indicates that the corresponding character from the name of the directory being renamed will be used, thus allowing an ambiguous rename. Thus `'*'` in the second filename, which is just equivalent to a series of `'?'`s, indicates that the whole of the filename or extension of the directory name will remain unchanged.

  If for some reason a particular directory cannot be renamed (eg. if a file or directory of the new name already exists) then the offending directory name will be printed along with an error message and the rename operation will continue with the next directory. If many errors occur, then `/P` will cause the output to pause at the end of the screen.

- Examples

  ```
  RNDIR UTIL COM
  ```

  The directory called `UTIL` in the current directory of the current drive is renamed `COM`.

  ```
  RNDIR A:\*.*/H/P *.OLD
  UTIL -- Duplicate filename
  ```

All directories, hidden and not hidden, in the root directory of drive `A:` are renamed with an extension of `.OLD`. The directory `UTIL` could not be renamed because a directory called `UTIL.OLD` already existed, so the error was printed. If many such errors were printed then `/P` would cause a prompt to be printed at the end of every screen full.

```
RNDIR COM + BAT *.OLD
```

The directories `COM` and `BAT` are renamed to `COM.OLD` and `BAT.OLD` respectively.

1.4.38 SET

- Format

  ```
  SET [name] [separator] [value]
  ```

- Purpose

  Displays/sets environment items.

- Use

  If no parameters are given, then all currently defined environment items and their current values are displayed. Initially there are several items set up to default values (see section 1.7 on Environment Items).

  If just a name is given as the parameter, then the current value of the specified environment item is printed.

  If the name is followed by a separator, then the separator is ignored and the name is set to the following value. If the value is blank (ie. not given) then the environment item is deleted from the environment space.

  The area of memory used for environment items is also used for disk buffers. Thus if a `'not enough memory'` error occurs when using the `SET` command, then it may help to reduce the number of disk buffers (see the `BUFFERS` command).

  Section 1.7 contains more information about environment items and the items and values that are set up by default.

- Examples

  ```
  A>SET
  ECHO=OFF
  PROMPT=OFF
  PATH=;
  TIME=12
  DATE=yy-mm-dd
  HELP=A:\HELP
  SHELL=A:\COMMAND2.COM
  ```

No parameters were given, so all the currently set environment items were printed, in this case typical default values.

```
SET HELP=A:\HELP
```

An item called HELP is set to the value A:\HELP.

```
A>SET HELP
A:\HELP
```

The current value of HELP is printed.

```
SET HELP=
```

The item HELP is set to a null value, thus removing it from the environment item list.

1.4.39 TIME

- Format

  ```
  TIME [time]
  ```

- Purpose

  Displays or sets the current time.

- Use

  If the time is given after the command, then the time is set to this value (for the format see below). If the time is not given after the command, then the current time is printed and the new time is prompted for and input. If no input is given (ie. if the 'enter' key alone is pressed) then the current time is not altered. Otherwise the input is assumed to be a new time, and is interpreted as described below. If the time is invalid then an error message is displayed and the new time again prompted for and input.

  The time is expected to consist of up to four numbers, separated by one of the following characters:

  ```
  SPACE TAB , - . / :
  ```

  with spaces allowed either side of the character. Any missing numbers will default to the current setting. The first number is the hour, the second is the minutes, the third is the seconds and the forth is the centi-seconds. The centi-seconds are not printed however since it is not very useful to know the current value, or indeed to enter a new one.

  The format in which the time is printed is flexible and can be changed. An environment item (see section 1.7 on Environment Items) called TIME

is set up by default to the value '12', which indicates that the time will be printed in 12 hour format with a following 'a' or 'p' for am and pm. The command SET TIME 24 will cause the time to be printed in 24 hour mode. The time can be input unambiguously in either format. The time format also affects the times printed by the DIR command.

- Examples

```
TIME 16:45
```

The current time is set to 4:45 pm.

```
A>TIME
Current time is 10:45:00a
Enter new time:
```

No parameters were given, so the current time is printed (in this case in 12 hour mode) and the new time prompted for.

```
TIME 10-50-30-23
```

The time is set to 30.23 seconds after 10:50 am.

1.4.40 TYPE

- Format

```
TYPE [/H] [/P] [/B] compound-filespec | device
```

- Purpose

Displays data from a file or device.

- Use

The compound-filespec specifies the files that are to be displayed, and /H allows hidden files to be typed. If the compound-filespec is ambiguous, then the filename is printed before each one is typed.

If /B is specified, then data is read from each file and displayed without modification on the screen, until the end of file is reached. This may have strange effect on the screen if the file contains control characters.

If /B is not given, then TYPE will look for the end-of-file character (CTRL-Z) and stop when it finds it. Also control characters except carriage return, line feed and tab will be converted into characters that can be printed, A for ^A, W for ^W, etc.

If /P is given, then the output will pause at the end of the screen until a key is pressed.

- Examples

  ```
  TYPE FILE1
  ```

  Data is read from the file and printed on the screen, up to the first end-of-file character.

  ```
  TYPE *.BAT /H/P
  ```

  All batch files, including hidden ones, are read in and displayed. A prompt is printed at the end of every screen full.

  ```
  TYPE AUTOEXEC.BAT + REBOOT.BAT
  ```

  The files AUTOEXEC.BAT and REBOOT.BAT are displayed.

  ```
  TYPE /B DIR1
  ```

  All files in the directory `DIR1` are printed on the screen and no interpretation is put on the data in the files.

1.4.41 UNDEL

- Format

  ```
  UNDEL [filespec]
  ```

- Purpose

  Recovers a previously deleted file.

- Use

  The `filespec` specifies which files are to be undeleted if possible, and defaults to `*.*`.

  Files can only be undeleted if they have been deleted using MSX-DOS 2 on an MSX-DOS 2 formatted disk and if no disk allocation has taken place since the file was originally deleted, which usually means that they have to be undeleted immediately after they have been deleted.

  Each deleted file and directory reference that is found in the directory specified by the filespec will be undeleted if its name is matched by the filename in the `filespec`, and if undeletion is possible. `UNDEL` can therefore be used to restore a directory removed with the `RD` or `RMDIR` commands; to restore the contents of the directory a further `UNDEL` command is required specifying the now undeleted directory.

  Note that `UNDEL` is a transient command, and therefore must be loaded from disk.

- Examples

```
UNDEL B:HELP.MAC
```

Attempts to undelete the file `HELP.MAC` from the current directory of drive `B:`.

```
UNDEL A:\DIR1
```

All undeletable files and directories in `DIR1` are undeleted.

1.4.42 VER

- Format

  ```
  VER
  ```

- Purpose

  Displays the system's version numbers.

- Use

  The version numbers of the three main components of the MSX-DOS disk system are displayed. Each version number consists of three digits. The first digit is the main MSX-DOS version number and for MSX-DOS 2 will always be 2. The second digit is the version number and will change for future versions that have, for example, had extra major features added. The last digit is the release number and will change with different releases of the same version of the system which have had minor changes, improvements and corrections made.

- Examples

  ```
  A>VER
  MSX-DOS kernel version 2.20
  MSXDOS2.SYS version 2.20
  COMMAND2.COM version 2.20
  Copyright 1988 ASCII Corporation
  ```

  The version numbers of all the components of the MSX-DOS disk system are printed out.

1.4.43 VERIFY

- Format

  ```
  VERIFY [ON | OFF]
  ```

- Purpose

  Displays/sets the current disk write verify state.

- Use

  If no parameters are given, then the current verify state is displayed on the screen.

  If `ON` or `OFF` is given, then the verify state is changed appropriately.

  The verify state affects all writes to disk. If `OFF`, the default state, then data is simply written. If `ON`, then after writing the data it is read back and compared with the original to ensure that it was written correctly. The extra overhead of this means that writing is slower when verify is on.

  This feature depends on the device driver, so this will have no effect if the driver does not have the feature.

- Examples

  ```
  A>VERIFY
  VERIFY=OFF
  ```

  No parameters were given, so the current verify state is printed, which in this case is off.

  ```
  VERIFY ON
  ```

  Disk write verification is turned on.

1.4.44 VOL

- Format

  ```
  VOL [d:] [volname]
  ```

- Purpose

  Displays or changes the volume name on a disk.

- Use

  If no parameters are given, or if only a drive name is given, then the volume name of the current or specified drive is printed.

  If a `volname` is given, then the volume name of the specified or current drive is changed to the specified volume name.

- Examples

  ```
  A>VOL B:
  Volume in drive B: has no name
  ```

  Just a drive was given, so the volume name for the disk in that drive is printed. In this case there was no volume name defined.

  ```
  VOL B:BACKUP
  ```

  The volume name of the disk in drive `B:` is changed to `BACKUP`.

1.4.45 XCOPY

- Format

```
XCOPY [filespec [filespec]] [options]
```

- Purpose

  Copies files and directories from one disk to another.

- Use

  The options available are:

```
[/A] [/E] [/H] [/M] [/P] [/S] [/T] [/W] [/V]
```

  XCOPY is an extended file copying command (compare with the COPY command) that can selectively copy both files and directories. The first filespec specifies the source filename, and if /H is given then hidden files will also be copied. The second filespec is the destination filename. Thus files can be renamed during the copy (as in the standard COPY command).

  /T (time) will cause the copied files to have the current date and time rather than the source file's date and time.

  If /A (archive) is specified, then only files with the 'archive' attribute set are copied. A file has an archive attribute in the same way as a 'hidden' attribute and a 'read only' attribute. It is set whenever a file is updated (written to).

  /M is similar to /A, but resets the archive bit after copying the file. Thus, using this option, files can be regularly copied onto another disk only if they have been updated, providing a file backup facility.

  /S causes XCOPY to copy directories as well as files. Within each directory, all files are copied and then any matching files within each directory are copied, with the directory being created on the destination if it does not already exist. Normally, these directories will not be created if no files are to be copied into them.

  /E can be given to cause the /S option to create all directories, even if they are empty.

  The /P (pause) option will cause XCOPY to pause and prompt before copying each file, which allows files to be selectively copied.

  /W (wait) causes XCOPY to pause and prompt before copying any files, so that disks can be changed.

  /V option can be given to turn write verification on for the duration of the XCOPY command (see the VERIFY command). This will ensure that data is written correctly to disks if the device driver being used has the feature, but will slow the operation down.

  Note that XCOPY is a transient command, and so must be loaded off disk.

- Examples

```
XCOPY B:\
```

All files in the root directory of drive `B:` are copied to the current directory of the current drive. There is no advantage in this case over using the standard built-in `COPY` command.

```
XCOPY *.* B: /H/S/M
```

All files, including hidden files, are copied to drive `B:` only if they have been modified since a similar command was last given. The archive attributes are then reset so that the files are marked as unmodified. Not only are all the files in the current directory copied, but so are directories and all their descendant directories and files.

1.4.46 XDIR

- Format

```
XDIR [filespec] [/H]
```

- Purpose

Lists all files within directories.

- Use

The `filespec` specifies which files are to be listed, and `/H` allows hidden files to be included.

`XDIR` is similar to the `DIR` command, but does not print the files' dates and times.

After all files in the specified directory have been listed, then files within descendant directories are also listed, and are shown indented. This allows a `DIR` of a complete directory tree or disk to be obtained.

Note that `XDIR` is a transient command, and so must be loaded off disk.

- Examples

```
A>XDIR
Volume in drive A: is MSX-DOS 2
X-Directory of A:\

MSXDOS2.SYS     r     4480
COMMAND2.COM    r    14976
AUTOEXEC.BAT            57
REBOOT.BAT              57
```

```
\UTILS
        CHKDSK.COM       7680
        DISKCOPY.COM     7168
        FIXDISK.COM       768
        UNDEL.COM        3968
        XCOPY.COM       10112
        XDIR.COM         7168
        MKSYS.BAT         569
        AUTOEXEC.BAT       47
        REBOOT.BAT         90
\HELP
        ASSIGN.HLP        819
        ATDIR.HLP        1527
        ATTRIB.HLP       1828
        .
        .
        .
292K in 117 files 530K free
```

The directory of the entire disk in or descending from the current directory of the current drive is printed.

```
XDIR B:\DIR1
```

All the files and directories and their contents are printed from the directory `DIR1`.

```
XDIR \*.COM/H
```

The names of all files, including hidden files, matching `*.COM` are displayed.

1.5 Redirecting and piping

`COMMAND2.COM` offers the redirection and piping features as described below. They may be bypassed by setting the environment item `"REDIR"` to `"OFF"` (`"SET REDIR=OFF"`), so the compatibility to MSX-DOS 1 or CP/M can be achieved.

1.5.1 Redirection

Most commands, CP/M programs and MSX-DOS programs output text to the screen by writing to the 'standard output', and read from the keyboard by reading from the 'standard input'. `COMMAND2.COM`, however, provides facilities for changing the standard input and standard output for the duration of the command to refer to other MSX-DOS devices or to files on disk by including one or more of the redirection symbols <, > and >> on the command line, followed by a filename.

For example, the ECHO command normally just outputs it's parameters to the screen by writing the characters to the standard output. It can be made to output to the printer instead by redirecting it's output, as follows:

```
ECHO text >PRN
```

which changes the standard output to refer to the device PRN for the duration of the ECHO command. Similarly, the command:

```
ECHO text >file1
```

will cause the specified file ("file1") to be created, and the output of the ECHO command written to the file. To append the output of a command to the end of an existing file, the > > symbol can be used instead of the > symbol, and the file will only be created if it does not already exist.

To change the standard input, the < symbol is used in a similar manner to the > symbol. In this case, the file must already exist, and must contain adequate input for the command. If the command attempts to read input beyond the end of the file, then it will be aborted since it cannot continue.

When redirection information is given on the command line, it is used by COMMAND2.COM to set up the redirection and then removed from the command line. Thus in the examples above, the ECHO command will not echo the redirection symbols or the filename.

If the input or output of a batch file is redirected, then that redirection is applied to all commands within the batch file. Individual commands within the batch file may still be redirected, however, which will override the batch file redirection. See section 1.6 on Batch Files for more information on commands in batch files.

1.5.2 Piping

As well as redirecting the input and output of a command or program to another device or a disk file, it is possible to redirect or 'pipe' the standard output of one command into the standard input of another. Typically the second command will be a program which reads from it's standard input, modifies the data, and writes it to it's standard output. Such a program is called a 'filter'. For example, a filter could be produced which read data from it's standard input, sorted it into alphabetical order, and wrote it to it's standard output. Thus the output of the DIR command could be sorted.

Piping is indicated on the command line by separating the two commands by the | symbol. The command to the left of the | symbol will be performed first, and it's output will be redirected to a temporary file created by COMMAND2.COM. Then the second command will be performed, with it's standard input redirected from the same temporary file. When the second command ends, the temporary file will be deleted. The standard output of the second command may of course have been piped into the standard input of a third command, and so on.

If any input redirection occurs on a command line involving a pipe, then the redirection is applied to the first command in the pipe, as all the other commands receive their standard input from the standard output of the previous command in the pipe. Similarly, if any output redirection occurs on a command line involving a pipe, then the redirection will apply to the last command on the command line.

It is not possible to use pipes on either the input or the output of batch files directly. It is, however, possible to use piping with batch files if they are executed with the COMMAND2 command (see section 1.4) since it is then the COMMAND2 command that is being redirected and not the batch file.

As mentioned above, in order to pipe the output of one command into the input of another, temporary files will be created and deleted by COMMAND2.COM. The location of these temporary files is specified by the TEMP environment item (see section 1.7 on Environment Items), and this may be changed to refer to any drive and directory (for example piping will be speeded up considerably if TEMP refers to a directory on a RAM disk). By default, TEMP refers to the root directory of the boot disk. The filename used for the temporary file is created by COMMAND2.COM, so TEMP should specify just the drive and directory. The filename is of the form:

```
%PIPExxx.$$$
```

where xxx is a three digit number chosen by COMMAND2.COM to avoid clashes with any other files in the TEMP directory.

1.6 Batch files

When a command is given to MSX-DOS and it is not one of the internal commands, a file of that name is searched for with an extension of COM or BAT. If not found in the current directory then the current search path is searched (see the PATH command). If a COM file is found, then it is loaded and executed. If a BAT file is found, then MSX-DOS starts execution of the batch file.

A batch file is a text file that contains a list of commands, and these commands are read from the file a line at a time and executed as though they were typed at the keyboard. Several of the commands described in section 1.4 are in fact provided mainly for use in batch files, such as ECHO and PAUSE.

As each command is read, normally it is executed immediately. An environment item ECHO exists, however, that can be set to ON (with the command SET ECHO ON) to cause each command line to be printed on the screen before it is executed (see section 1.7 for Environment Items). The command line is echoed in this way after % parameter substitution (see below) has taken place. The command SET ECHO OFF will restore the normal state.

In the command line that invoked the batch file, parameters may follow the name of the batch file just like any other command or transient program name. These parameters may be accessed anywhere in the batch file by specifying %0 to %9. %1 is the first parameter specified in the command line, %2 is the second

and so on. `%0` is the name of the batch file itself. The `%` number will be replaced by the parameter on the original command line, and may appear anywhere on a batch file command line. To actually use a `%` symbol on a command line a double `%` must be given (`'%%'`) which will then be replaced by a single one.

If the execution of any command in a batch file is terminated prematurely for some reason (typically the `CTRL-STOP` or `CTRL-C` key being pressed) then the following prompt is issued:

```
Terminate batch file (Y/N)?
```

If the response to this is `'Y'`, then execution of the whole batch file is stopped. If the response is `'N'`, then batch file execution continues with the next command in the batch file.

After MSX-DOS has executed a command in batch file, it may need to read the next command in the batch file off disk. If the correct disk is not in the drive when it comes to do this, then a prompt is issued. For example, the following prompt will be issued if the batch file was originally executed from drive `A:`:

```
Insert disk for batch file in drive A:
Press any key to continue...
```

When the correct disk has been inserted and a key pressed, batch file execution will continue normally.

Below is a very simple batch file, which just prints the first few parameters.

```
ECHO Parameter 0 = %0
ECHO Parameter 1 = %1
ECHO Parameter 2 = %2
ECHO Parameter 3 = %3
```

If this is called `MYBAT.BAT` then the command `MYBAT a b c` will output:

```
Parameter 0 = MYBAT
Parameter 1 = a
Parameter 2 = b
Parameter 3 = c
```

When MSX-DOS starts up for the first time, a special batch file called `AUTOEXEC.BAT` is looked for and is executed if found. This may contain any MSX-DOS command, and usually contains once-only initialization commands, such as a `RAMDISK` command to set up a RAM disk.

One `%` parameter is passed to `AUTOEXEC.BAT` as `%1`. This the drive that MSX-DOS booted from and is in the form of a normal drive letter followed by a colon.

Another special batch file is `REBOOT.BAT`. This is executed when MSX-DOS is re-booted after using DISK-BASIC. As with `AUTOEXEC.BAT` files, the single `%1` parameter passed is the drive from which MSX-DOS was re-booted.

Usually some commands need to be performed whenever MSX-DOS is booted, whether for the first time or sometime later, and these are put in the REBOOT batch file. They can then be executed from the AUTOEXEC batch file by ending it with the command REBOOT %1. An example of a command that might be put in the REBOOT batch file is the PATH command to set up the transient command search path. When setting up the search path using the command, %1 can be used to set up the path on whatever drive was booted from.

When a command in a batch file is the name of another batch file, then that second batch file is executed in the normal way. When it ends, control passes back to the interactive command interpreter, and not to the first batch file. Batch file commands thus 'chain' together.

If it is desired to 'nest' batch files ie. to pass control back to the first batch file above, then this can be done with the COMMAND2 command (see section 1.4), passing the name of the second batch file as the parameter. Then when the second batch file ends, the first one will be continued with the command after the COMMAND2 command.

A typical AUTOEXEC batch file is as follows:

```
ECHO AUTOEXEC executing
RAMDISK 100
RAMDISK
COPY COMMAND2.COM H:\
REBOOT %1
```

A typical REBOOT batch file is as follows:

```
ECHO REBOOT executing
PATH H:\, %1\UTILS, %1\BATCH
SET SHELL=H:\COMMAND2.COM
SET TEMP=H:\
SET PROMPT ON
H:
```

When the AUTOEXEC batch file executes, the message "AUTOEXEC executing" is printed, and then a RAM disk is set up with a maximum size of 100K. Another RAMDISK command is then given which will print out the actual size of RAM disk created. The COPY command then copies COMMAND2.COM onto the RAM disk so that it can load and re-load quickly. Finally the REBOOT batch file is executed, with the %1 parameter (the boot drive) passed to it.

The REBOOT batch file prints a message and then sets a PATH. The first item in the path refers to the RAM disk that was created by the AUTOEXEC batch file, and the other items refer to directories on the disk from which MSX-DOS was booted (ie. %1). The SHELL environment item is set up so that COMMAND2.COM can re-load quickly off the RAM disk, and the TEMP environment item is set up so that temporary piping files are created on the RAM disk. The prompt is set ON so that the current directory is printed as the prompt and, finally, the RAM disk is made the default drive.

1.7 Environment items

MSX-DOS maintains a list of 'environment items' in its work area. An environment item is a named item that has a value associated with it.

An environment item can have any name chosen by the user, and can consist of the same characters that are valid in a filename. The maximum length of an environment item name is 255 characters. MSX-DOS provides several environment items set up by default.

The value of an environment item is simply a string of arbitrary characters up to a maximum length of 255. No processing is performed on the characters and so the casing of characters is preserved. Any environment item that does not exist has a null value (ie. no characters).

An environment item can be changed or set up by the `SET` command, which can also display currently set environment items.

The environment items set up by default and the manner in which their value is interpreted are as follows:

1.7.1 ECHO

This controls the echoing of lines read from a batch file (see section 1.6 on Batch Files). Any value except `'ON'` (lowercase also allowed) is interpreted as `'OFF'`.

1.7.2 PROMPT

This controls the displaying of the prompt at command level. Any value except `'ON'` is interpreted as `'OFF'`.

When `PROMPT` is `OFF`, as it is by default, then the prompt consists of the current drive followed by `'>'` eg. `A>`.

When `PROMPT` is `ON`, then the prompt consists of the current drive and the current directory of that drive followed by `'>'` eg. `A:\COM>`. In order to do this, the current drive must be accessed to read the current directory and so may take a little longer to appear.

1.7.3 PATH

The current search path by which `COMMAND2.COM` searches the command given is maintained as an environment item `PATH`, and it is this that the `PATH` command manipulates.

1.7.4 SHELL

The `SHELL` environment item indicates where the command interpreter (`COMMAND2.COM`) exists, and is set up by default to where it was loaded from. When the command interpreter needs to re-load itself from disk (after running a transient command) it looks at the `SHELL` environment item and attempts to load itself from the file

that it specifies. If this gives an error then it attempts to load itself from the root directory of the drive that it was originally loaded from.

To cause the command interpreter to re-load itself from another drive or directory, `COMMAND2.COM` can be copied there and `SHELL` set to refer to it. For example, it might be copied to the `RAMDISK` with the command `COPY COMMAND2.COM H:\` and then `SHELL` set with the command `SET SHELL=H:\COMMAND2.COM`.

1.7.5 TIME

`TIME` specifies the format the time is displayed by MSX-DOS. If not `'24'`, which indicates that it is displayed as a 24-hour time, then `'12'` is assumed, which indicates that it is displayed as a 12-hour time with an am. or pm. indication. The `TIME` environment item does not apply when the time is input because it can be input in either format unambiguously.

1.7.6 DATE

`DATE` specifies the format the date is displayed and input by MSX-DOS. It defaults to a format appropriate for the country of origin of the MSX machine. It takes the form of three letters or three letter pairs separated by date/time separators (see the `DATE` command). To set the American format, for example, the command `SET DATE=MM/DD/YY` could be given.

1.7.7 HELP

When the `HELP` command is given the name of command for which help is required, then it reads the information displayed from a file on disk. This file is in the directory specified by the `HELP` environment item. It defaults to a directory called `HELP` in the root directory of the drive that MSX-DOS was booted from.

1.7.8 APPEND

`APPEND` is not actually defined by default, but when set up is an environment item that has a special meaning to the system. It is used only with standard CP/M programs.

CP/M programs do not know how to use sub-directories because CP/M does not have sub-directories, but instead just has the equivalent of the current directory. When such a program opens a file, it searches for it only within this single directory and thus only has drives and filenames, not paths.

When a CP/M program is run under MSX-DOS and attempts to open a file, it only searches for the filename in the current directory of the appropriate drive. Similarly, when the user types in a filename to a CP/M program it may only contain a drive and filename, and thus also refers only to files in the current directory.

When this search is performed through MSX-DOS, if the file is not found in the current directory, then the APPEND environment item is looked at. If it is not set up then the file has not been found. If set up, then it is assumed to be a path name, and specifies a single alternative directory in which the search for the file continues.

This will only be of use if the CP/M program opens a file and then reads or writes to it. If it attempts to, for example, delete a file or create a file, then APPEND will not be used. Indeed, it may have undesirable effects and for this reason it is recommended that APPEND is used normally only in a batch file which sets it up, executes the CP/M program, and then unsets it again.

Typical uses for APPEND include specifying the directory in which large programs (such as word processors and database programs) find *overlay* or messages files, and specifying the directory in which compilers, assemblers and linkers find their source and working files. Typical cases for which APPEND is not suitable and may have undesirable effects include editing a file with a wordprocessor when the file to be edited probably cannot be in a directory other than the current one, even if APPEND is set up.

1.7.9 PROGRAM and PARAMETERS

These special environment items are set up by COMMAND2.COM when a transient command is executed and removed when it finishes, and should thus be avoided for general use.

1.7.10 TEMP

When piping is performed (see section 1.5 on Redirection and Piping) it is necessary for COMMAND2.COM to create one or more temporary files, and the TEMP environment item indicates the drive and directory in which these temporary files are to be created. By default, it refers to the root directory of the boot drive, and typically may be changed to refer to a RAM disk since this increases the speed of piping.

Although the standard MSX-DOS system only uses TEMP for piping, any other programs and utilities that need to create temporary files may also use the TEMP environment item.

1.7.11 UPPER

UPPER controls whether the command line from 80h to be passed to the transient program be converted to uppercase. Any value except 'ON' is interpreted as 'OFF'.

When UPPER is 'OFF' (default), no conversion will be done and the values will be passed to the transient program as they are typed.

When UPPER is 'ON', each character in the command line will be converted to its associated uppercase character and then passed to the transient program. This is compatible to CP/M environment.

1.7.12 REDIR

REDIR controls whether the redirection or piping characters in the command line be processed by COMMAND2.COM. Any value except 'ON' is interpreted as 'OFF'.

When REDIR is 'OFF', the redirection or piping characters will be passed to the transient program as they are typed, and the transient program may process them.

When REDIR is 'ON' (default), the redirection or piping characters will be interpreted and executed by COMMAND2.COM, so they will not be passed to the transient program.

1.8 Errors and messages

1.8.1 Disk errors

Disk errors occur when a command or program is attempting to access a disk and fails for some reason, such as a disk not being in the drive. When this happens, message and prompt appears which allows the user the either retry the operation which may now work (eg. if a disk has been inserted into the drive), to ignore the operation or to abort the entire command.

An example disk error message and prompt is as follows, and may be given if the disk was taken out whilst drive A: was being accessed:

```
Not ready reading drive A:
Abort, Retry or Ignore (A/R/I)?
```

The 'not ready' part of the message indicates why the disk operation failed, and other possibilities exist (see below). 'reading' indicates that the command was trying to read the disk, and may be replaced by 'writing' if that is what it was doing. 'drive A:' is the drive in which the disk was attempted to be accessed.

The 'Abort, Retry or Ignore' part indicates the possible actions that can be taken by the user, and these are selected by pressing the 'A', 'R' or 'I' key.

If *Abort* is selected, then the entire command is aborted and the message 'Disk operation aborted' is printed before another command can be typed.

If *Retry* is selected then the failed disk operation will simply be retried and may fail again or may work, particularly if some corrective action has been taken such as inserting a missing disk.

Ignore causes the failed disk operation to be ignored by the command. In many cases, ignoring an error may not be recommended and in these cases the *Ignore* option will not even be displayed, although it may still be selected. Doing so may however cause serious system malfunction and could destroy data on the disk. Even if the *Ignore* option is displayed, it should be used with extreme caution, and only when all else fails. Normally *Ignore* is only used when the data on a disk has got corrupted and ignoring the disk error offers the only possibility of recovering all or part of the data.

A few serious errors which generally mean the disk has been corrupted beyond possible use are automatically aborted, and just the appropriate error message is printed (eg. `'Bad file allocation table'`).

The possible errors that can occur as disk errors and their meanings are, in alphabetical order, as follows:

`Bad file allocation table`

The disk contains an invalid file allocation table (FAT). The FAT is an area on disk in which the system keeps information to tell it where on the disk the data in each file is stored. Thus if the FAT is invalid, it cannot read any data at all. This message usually means that the disk has been corrupted beyond possible use.

`Cannot format this drive`

An attempt was made to format a disk in a drive that does not support disk formatting. This probably means that a `FORMAT` command was given specifying the RAM disk.

`Data error`

The data was read or written without error, but the CRC check failed. This usually means the disk has been corrupted.

`Disk error`

The data could not be read or written to the disk.

`Incompatible disk`

An attempt was made to access 2D or 1D disk or a double sided disk in a single sided drive.

`Not a DOS disk`

The disk is not a format that MSX-DOS can read. For example, although MSX-DOS can execute CP/M programs it cannot access CP/M disks.

`Not ready`

The disk is not in the drive being accessed. The disk should be inserted into the drive and `'Retry'` selected.

`Sector not found`

MSX-DOS tried to read or write to a non-existent sector. May indicate that the disk has been corrupted.

`Seek error`

The requested track on the disk could not be found. Could mean a corrupted disk or faulty disk drive.

`Unformatted disk`

The disk has not been formatted. Use the `FORMAT` command on the disk before accessing it.

`Verify error`

Only occurs when verify is on, and means that data appeared to be written to disk successfully but when read back was found to be different to that written.

`Write error`

Data was not written correctly.

`Write protected disk`

The disk is write protected and an attempt was made to write data to it. The disk should be made unprotected and `'Retry'` selected.

`Wrong disk and Wrong disk for file`

MSX-DOS accessed a disk once and subsequently needed to access it again, but found that the drive contained a different one. The correct disk should be inserted and `'Retry'` selected.

1.8.2 Command errors

Command errors occur when a command cannot perform its intended function for some reason.

If an error occurs in a command and it is unlikely to be able to continue, then an appropriate error message is printed, and the next command is read at the prompt.

An example error message is:

```
*** File not found
```

The three asterisks `***` are printed first to indicate that an error has occurred. The message is then printed, followed by the normal command prompt on the next line. The possible errors that can occur are listed below.

When a command error occurs in a specific situation, an 'error type' message may also be printed. For example, generally when a required file cannot be found on disk, the `'File not found'` message is printed as in the above example. However, if the file required is a file specified by the redirection symbol < (see section 1.5 on Redirection and Piping) then the message printed will be:

```
*** Redirection error: File not found
```

The possible error types are:

- `Batch file error`

  An error occurred whilst attempting to read from a batch file, for example a disk error occurred and *'abort'* was selected.

- `Piping error`

  The error occurred during a piping operation, probably in connection with the temporary files that `COMMAND2.COM` creates (see section 1.5 on Redirection and Piping). For example, the `TEMP` environment item did not refer to a valid drive or directory (see section 1.7 on Environment Items).

- `Redirection error`

  The error occurred during a redirection operation. For example, an invalid filename was specified after a redirection symbol <, > or >>, or the specified input file was not found (see section 1.5 on Redirection and Piping).

- `Standard input error`

  An error occurred on the standard input to a command or program after redirection or piping has been set up, for example the standard input has been redirected from a file and the end of the file has been reached.

- `Standard output error`

  An error occurred on the standard output of a command or program after redirection or piping has been set up, for example the standard output has been redirected to a file and the disk is full.

  Many commands operate on files or directories, and if an ambiguous filename is given then the command operates on several files or directories (for example the RENAME command or the COPY command). Often an error occurs when it is trying to perform the command on one of the files, but which may be successful on one of the other files. In this case, the filename is printed followed by the error message and the command then continues. For example:

  ```
  COMMAND2.COM -- File cannot be copied onto itself
  ```

The possible command errors that can be given are, in alphabetical order, as follows:

`Cannot concatenate destination file`

This error is given by `CONCAT` and means that one of the filenames matched by the source file specification is the destination file. This is not necessarily wrong but may indicate a mistake in the command.

`Cannot create destination file`

This is given by `COPY`, and usually means that the destination file for the file it is copying would, if it was created, overwrite a file that was already in use. This is likely to be a previously copied source file but may be some other file such as the currently executing batch file.

`Cannot overwrite previous destination file`

This is given by `COPY`, and means that the destination file for the file it is copying would, if it was created, overwrite the destination file of the file that was previously copied. This usually means that the intended destination was a directory but that its name was misspelled.

`Cannot transfer above 64K`

This should not normally occur from commands.

`Command too long`

A command that was given is too long. This will not occur when typing commands from the keyboard, but may occur from a batch file. The maximum length of a command is 127 characters after `%` parameter substitution.

`Ctrl-C pressed`

The command was interrupted by pressing `CTRL-C`.

`Ctrl-STOP pressed`

The command was interrupted by pressing `CTRL-STOP`.

`Directory exists`

A command attempted to create a new file or directory on disk with the same name as an existing directory.

`Directory not empty`

The `RMDIR` (`RD`) command tried to remove a directory that contains files or other directories. These must be deleted first with the `ERASE` and `RMDIR` commands since directories must be empty before they can be removed.

`Directory not found`

A directory command (eg. `RNDIR`) could not find the specified directory.

`Disk full`

There is no more room on the disk and files will have to be deleted and the command given again.

`Disk operation aborted`

A disk error occurred and the *'Abort'* option was chosen, thus aborting the whole command.

`Duplicate filename`

`RENAME` (`REN`) or `RNDIR` cannot perform the specified rename because the new filename is the same as a filename that already exists. Also occurs from `MOVE` or `MVDIR` because a filename already exists in the destination directory with the same name as the file or directory being moved.

`End of file`

This should not normally occur from commands.

`Environment string too long`

This should not normally occur from commands.

`Error on standard input`

This should not normally occur from commands, and means that an error occurred while a command was attempting to read from the keyboard.

`Error on standard output`

This should not normally occur from commands, and means that an error occurred while a command was attempting to write to the screen.

`File access violation`

This should not normally occur from commands.

`File allocation error`

This should not normally occur from commands.

`File cannot be copied onto itself`

The destination file when trying to do a `COPY` is the same file as the source.

`File exists`

`MKDIR` (`MD`) tried to create a new directory but a file with the same name already exists in the specified directory.

`File for HELP not found`

The `HELP` command looked for a file to get the help text from but could not find it. Help files are usually kept in a directory called `\HELP` on the boot disk.

`File handle not open`

This should not normally occur from commands.

`File is already in use`

A command tried to modify a file that is already being used for some other purpose, such as the currently executing batch file.

`File not found`

A command could not find the specified file or files.

`Internal error`

This should not normally occur from commands.

`Invalid MSX-DOS call`

This should not normally occur from commands.

`Invalid attributes`

Usually means an invalid +/- attribute was specified in `ATTRIB` or `ATDIR`.

`Invalid date`

The date typed into the `DATE` command is not a valid date or was typed in in an invalid format.

`Invalid device operation`

A command cannot perform its function on one of the built-in system devices eg. a file cannot be renamed `CON`.

`Invalid directory move`

`MVDIR` attempted to move a directory into one of its own descendant directories, which cannot be done.

`Invalid drive`

A drive that does not exist was specified.

`Invalid environment string`

The name of an environment item contains invalid characters. Only those characters valid in filenames are valid in environment item names.

`Invalid file handle`

This should not normally occur from commands.

`Invalid filename`

A filename contains invalid characters. This may be a filename explicitly given, or may be the result of attempting to rename a file with an ambiguous new name.

`Invalid number`

A number given in a command contained non-digit characters.

`Invalid option`

An invalid letter was given after a / on a command line.

`Invalid . or .. operation`

A command cannot perform its function on the special '.' and '..' directories that are present at the start of all sub-directories.

`Invalid parameter`

The parameter to a command is generally not valid for that command in some way.

`Invalid pathname`

A path specified on a command line does not exist or is syntactically incorrect.

`Invalid process id`

This should not normally occur from commands.

`Invalid time`

The time typed into the `TIME` command is not a valid time or was typed in in an invalid format.

`Missing parameter`

A command expected a parameter but did not find one.

`No spare file handles`

Should not normally occur from commands.

`Not enough memory`

Not enough memory is available for the given command. For example, a large program to large to fit into memory or not enough memory for a new environment string.

`Not enough memory, system halted`

This special error message is printed when MSX-DOS attempts to start up and finds that there is not enough memory to continue. As the message suggests, the computer must then be reset. This should normally occur.

`Pathname too long`

A path is too long. Either the length of the pathname given exceeds 100 characters, or the total length of a path from the root directory to a file is more than 63 characters.

`RAM disk already exists`

Should not normally occur from commands.

`RAM disk does not exist`

The `RAMDISK` command was used to display the current size of the RAM disk, but no RAM disk exists.

`Read only file`

An attempt was made to modify or overwrite a file marked as read only. The `DIR` command shows this, and the `ATTRIB` command can make it not read only.

`Root directory full`

The fixed maximum number of files in the root directory (often 64 or 112) has been reached. Directories do not have this limitation.

`System file exists`

An attempt was made to create a file which would, if it was created, overwrite a file that is marked as a system file. System files are not used in MSX-DOS, and are not shown by the `DIR` command or accessible from any other commands, and so this error should not normally occur with commands.

`Too many parameters`

All the parameters a command expected were found on a command line, but there were still more parameters left on the end of the line.

`Unrecognized command`

A given command was not an internal command or an external `COM` or `BAT` command found along the current search path as set by the `PATH` command.

`Wrong version of command`

After executing a program, `COMMAND2.COM` tried to re-load itself from the `COMMAND2.COM` file on disk, and found it was not the same version. A prompt is then printed and `COMMAND2.COM` will attempt to re-load itself again.

`Wrong version of MSX-DOS, system halted`

This special error message is printed when MSX-DOS attempts to starts up and finds that some other part of the MSX-DOS system has a version number earlier than required. As the message suggests, the computer must then be reset. This should not normally occur.

Internally, errors are represented as error numbers. The numbers corresponding to the errors above start at 255 and decrease. If an error number is received for which there is no message, then it is printed. Numbers above 64 are reserved for future version of MSX-DOS and so are called *'system errors'* and numbers below 63 can be used by external commercially available programs and are called *'user errors'*. User errors below 32 never print a message. The two default error messages (which will not normally occur from commands) are thus:

```
System error 64
```

and

```
User error 63
```

where the 64 and 63 are example error numbers. The only command which uses error numbers is the *EXIT* command. A list of the actual numbers for the above messages is available in chapter 2.

1.8.3 Prompt messages

There are several situations in which user interaction is required before the system can continue with what it was doing, typically inserting a disk. Also many potentially dangerous commands require confirmation prompts to be answered before they perform their operation. These various system prompts are given below.

```
All data on drive A: will be destroyed
Press any key to continue...
```

This prompt is given by the FORMAT command, and is issued to reduce the risk of accidentally formatting the wrong disk. To abort the FORMAT command, CTRL-STOP or CTRL-C can be pressed.

```
Destroy all data on RAM disk (Y/N)?
```

A RAMDISK command was given to set up a RAM disk, but a RAM disk already existed. If the response to the prompt is 'Y', then any files on this existing RAM disk will be destroyed. A response of 'N' or CTRL-STOP or CTRL-C will abort the command.

```
Disk in drive A: will only be able to boot MSX-DOS
Press any key to continue...
```

This prompt is given by the FIXDISK command, and is issued to reduce the risk of accidentally updating a non-MSX-DOS 2 disk. To abort the FIXDISK command, CTRL-STOP or CTRL-C can be pressed.

```
Erase all files (Y/N)?
```

This prompt occurs when a DEL (or ERA or ERASE) command is given specifying all the files in a directory, and is issued to reduce the risk of accidentally deleting a lot of files.

```
Insert COMMAND2.COM disk in drive A:
Press any key to continue...
```

This may occur after running a program, and requires a disk containing COMMAND2.COM in the root directory to be present in the specified drive. After inserting the disk in the drive (which is the drive from which MSX-DOS was originally booted) and pressing a key, the system will continue as normal. If COMMAND2.COM has been copied somewhere else (eg. a RAM disk) then the SHELL environment item can be set up to make COMMAND2.COM re-load from there instead (see section 1.7 on Environment Items).

```
Insert batch file disk in drive A:
Press any key to continue...
```

This may occur during the execution of a batch file, and means that the system needed to read the next command from the batch file but found that the wrong disk was in the drive. After inserting the disk in the specified drive (which will be the drive from which the batch file was originally started) and pressing a key, execution of the batch file will continue as normal.

```
Press any key to continue...
```

This prompt is generally issued when some user interaction is required, and is normally printed after some other message which describes the action required. It is also printed by the `PAUSE` command. To abort the command that issued the prompt, `CTRL-STOP` or `CTRL-C` can be pressed.

```
Terminate batch file (Y/N)?
```

When MSX-DOS aborts a command prematurely (such as when the `CTRL-STOP` or `CTRL-C` key is pressed) and the command was executing in a batch file, this prompt is issued. If the response is `'Y'` then the batch file will also be aborted. If `'N'` is the response then the batch file will continue with the command that follows the aborted command.

1.9 Command summary

The following is a list of all the standard commands available in MSX-DOS, together with their syntax and purpose.

```
ASSIGN [d: [d:]]
```

Sets up the logical to physical translation of drives.

```
ATDIR +|-H [/H] [/P] compound-filespec
```

Changes the attributes of directories to make them hidden/not hidden.

```
ATTRIB +|- R|H [/H] [/P] compound-filespec
```

Changes the attributes of files to make them hidden/not hidden and read only/not read only.

```
BASIC [program]
```

Transfers control to MSX disk BASIC.

```
BUFFERS [number]
```

Displays or changes the number of disk buffers in the system.

```
CD [d:] [path]
```

Displays or changes the current directory.

```
CHDIR [d:] [path]
```

Displays or changes the current directory.

```
CHKDSK [d:] [/F]
```

Checks the integrity of the files on the disk.

```
CLS
```

Clears the screen.

```
COMMAND2 [command]
```

Invokes the command interpreter.

```
CONCAT [/H] [/P] [/B] [/V] compound-filespec filespec
```

Concatenates (joins together) files.

```
COPY [/A] [/H] [/T] [/V] [/P] compound-filespec [filespec]
```

Copies data from files or devices to other files or devices.

```
DATE [date]
```

Displays or sets the current date.

```
DEL [/H] [/P] compound-filespec
```

Deletes one or more files.

```
DIR [/H] [/W] [/P] [compound-filespec]
```

Displays the names of files on disk.

```
DISKCOPY [d: [d:]] [/X]
```

Copies one disk to another.

```
ECHO [text]
```

Prints text in a batch file.

```
ERA [/H] [/P] compound-filespec
```

Deletes one or more files.

```
ERASE [/H] [/P] compound-filespec
```

Deletes one or more files.

```
EXIT [number]
```

Exits COMMAND2.COM to the invoking program.

```
FIXDISK [d:] [/S]
```

Updates a disk to the full MSX-DOS 2 format.

```
FORMAT [d:]
```

Formats (initializes) a disk.

```
HELP [subject]
```

Provides on-line help for an MSX-DOS feature.

```
MD [d:] path
```

Creates a new sub-directory.

```
MKDIR [d:] path
```

Creates a new sub-directory.

```
MODE number
```

Changes the number of characters/line on the screen.

```
MOVE [/H] [/P] compound-filespec [path]
```

Moves files from one place to another on a disk.

```
MVDIR [/H] [/P] compound-filespec [path]
```

Moves directories from one place to another on a disk.

```
PATH [ [+|-] [d:]path [ [d:]path [ [d:]path ...]] ]
```

Displays or sets the COM and BAT command search path.

```
PAUSE [comment]
```

Prompts and waits for a key press in a batch file.

```
RAMDISK [number[K]] [/D]
```

Displays or sets the RAM disk size.

```
RD [/H] [/P] compound-filespec
```

Removes one or more sub-directories.

```
REM [comment]
```

Introduces a comment in a batch file.

```
REN [/H] [/P] compound-filespec filename
```

Renames one or more files.

```
RENAME [/H] [/P] compound-filespec filename
```

Renames one or more files.

```
RMDIR [/H] [/P] compound-filespec
```

Removes one or more sub-directories.

```
RNDIR [/H] [/P] compound-filespec filename
```

Renames one or more sub-directories.

```
SET [name] [separator] [value]
```

Displays or sets environment items.

```
TIME [time]
```

Displays or sets the current time.

```
TYPE [/H] [/P] [/B] compound-filespec | device
```

Displays data from a file or device.

```
UNDEL filespec
```

Recovers a previously deleted file.

```
VER
```

Displays the system's version numbers.

```
VERIFY [ON | OFF]
```

Displays/sets the current disk write verify state.

```
VOL [d:] [volname]
```

Displays or changes the volume name on a disk.

```
XCOPY [filespec [filespec]] [/A][/E][/H][/M][/P][/S][/T][/V][/W]
```

Copies files and directories from one disk to another.

```
XDIR [filespec] [/H]
```

Lists all files within directories.

1.10 DISK-BASIC 2.0

1.10.1 Overview

When the system disk (the one which contains MSXDOS2.SYS and COMMAND2.COM) does not exist at the system startup, or when MSX-DOS BASIC command is executed, DISK-BASIC 2.0 will be started.

DISK-BASIC 2.0 is the extended version of previous DISK-BASIC 1.0. Instructions to operate with the RAM disk or the directory has been added or extended.

1.10.2 Description of commands

CALL CHDIR

Format CALL CHDIR("[d:][path]")

Purpose Set or display current directory.

Example CALL CHDIR("WORK")

Description The function is the same as MSX-DOS CHDIR. See CHDIR.

CALL CHDRV

Format CALL CHDRV("[d:]")

Purpose Switch default drive.

Example CALL CHDRV("H:")

Description The same action is taken as when the drive name is given at MSX-DOS prompt.

CALL MKDIR

Format CALL MKDIR("[d:][path]")

Purpose Create new subdirectory.

Example CALL MKDIR("WORK")

Description The function is the same as MSX-DOS MKDIR. See MKDIR.

CALL RMDIR

Format CALL RMDIR("[d:][path]")

Purpose Remove one or more subdirectories.

Example CALL RMDIR("WORK")

Description The function is the same as MSX-DOS RMDIR. See RMDIR.

`CALL RAMDISK`

Format `CALL RAMDISK[([number][, variable name])]`

Purpose Set size of RAM disk or assign it into variable.

Example
```
CALL RAMDISK(32)
CALL RAMDISK(1000, A)
```

Description The function is the same as MSX-DOS `RAMDISK`. See `RAMDISK`.

`CALL SYSTEM`

Format `CALL SYSTEM` or `CALL SYSTEM[("DOS command name")]`

Purpose Pass control back to MSX-DOS.

Example `CALL SYSTEM("WORK")`

Description `CALL SYSTEM` passes the control back to MSX-DOS. The command name may be given to specify the operation to be executed after the control returns to `DOS`. If no command name is specified, `REBOOT.BAT` in the root directory on the boot drive, if any, will be executed.

`FILES`

Format `FILES["filename"][, L]`

Purpose Display names of files and directories on disk.

Example `FILES"W*.*"`

Description The function is the same as MSX-DOS `DIR`. See `DIR`. `"FILES, L"` displays the names in the long format.

Chapter 2

Programming environment

This chapter describes the interface to transient programs provided by MSX-DOS version 2.20.

2.1 Introduction

This chapter describes the environment which MSX-DOS 2 provides for transient programs on MSX 2 computers. It is intended as a guide for writing new programs to run under MSX-DOS 2 and also to assist in converting existing CP/M and MSX-DOS 1 programs to take advantage of the advanced features.

MSX-DOS 2 provides many enhancements to the standard CP/M and MSX-DOS 1 environment, but is largely compatible with existing programs. The main features include:

- MS-DOS style tree structured directories
- File handles
- Environment strings
- Proper error handling

Many extra DOS calls are implemented, and these are accessed via the DOS entry jump at address 5 (the 'BDOS' jump in CP/M). The descriptions of the individual functions can be found in chapter 3.

Throughout this manual, the term MSX-DOS is used generally to refer to MSX-DOS 2 unless otherwise stated.

2.2 Transient program environment

This chapter describes the environment in which transient programs are executed under MSX-DOS, including entry and exit to the program and memory usage.

2.2.1 Entry from MSX-DOS

A transient program will be loaded at address `0100h`, the start of the TPA (Transient Program Area), and is CALLed by MSX-DOS with the stack pointer set to the end of the TPA. If the stack pointer points to that location, as much RAM as possible can be used as the stack. If it is undesirable, then the transient program must set up its own stack in the TPA.

The contents of the Z80 registers when a transient program is entered are undefined. The first 256 bytes of RAM starting at the address 0 will have been set up with various parameters and code as described in section 2.2.3.

Interrupts are enabled when a transient program is entered and should generally be left enabled. MSX-DOS function calls will generally re-enable interrupts if the transient program has disabled them.

2.2.2 Return to MSX-DOS

A transient program can terminate itself in any of the following four ways:

1. Returning, with the original stack pointer.

2. Jump to location `0000h`.

3. MSX-DOS "Program Terminate" function call.

4. MSX-DOS "Terminate with Error Code" function call.

The first two of these methods are identical as far as MSX-DOS is concerned, and are compatible with CP/M and MSX-DOS 1. The third method is also compatible with CP/M and MSX-DOS 1 and is equivalent to doing a "Terminate with Error Code" function call with an error code of zero.

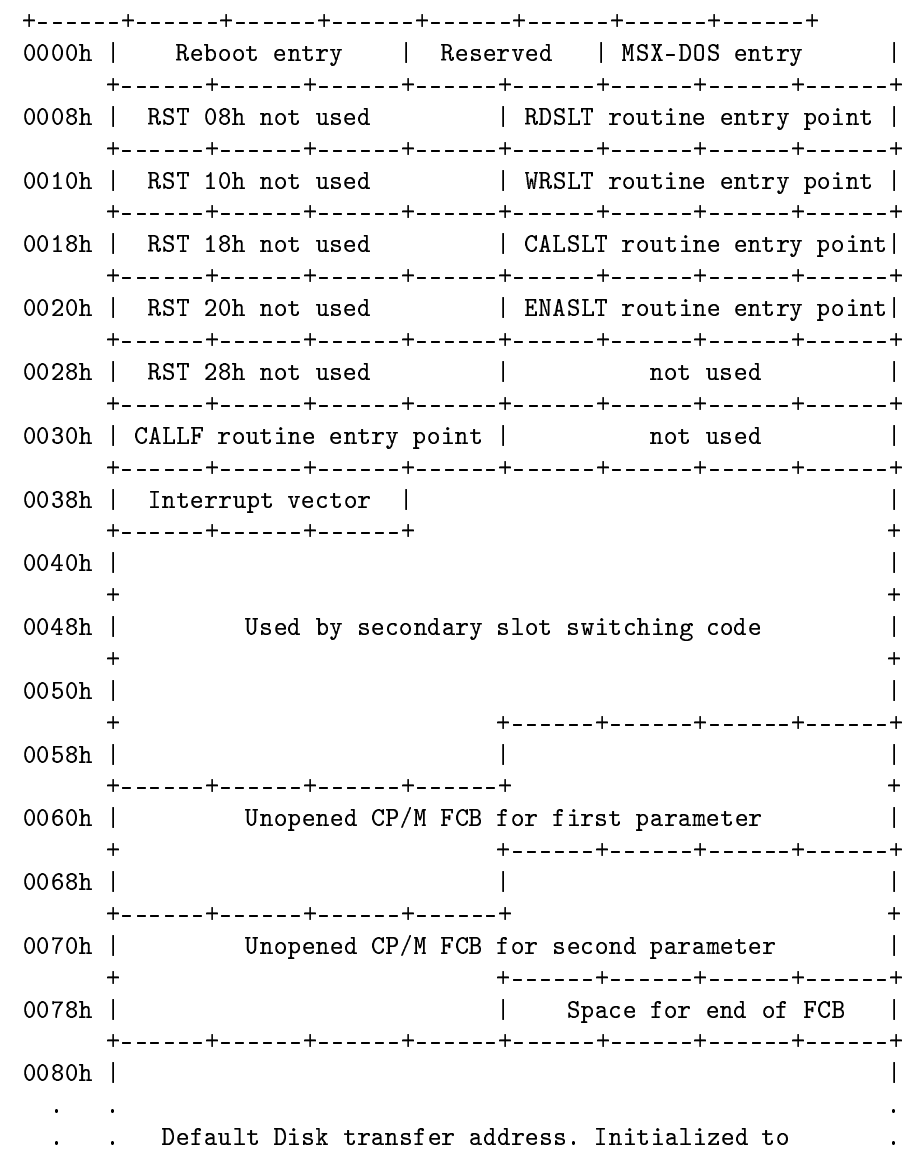
The new "Terminate with Error Code" function allows the program to return an error code to MSX-DOS, the first three terminating methods always returning an error code of zero (no error). All specially written programs and converted CP/M programs should use this new function, even for returning an error code of zero.

Various other events outside the control of a program can cause it to terminate. For example, typing `"CTRL-C"` or `"CTRL-STOP"` at the keyboard, by the user selecting "Abort" as the response to an "Abort/Retry/Ignore" disk error message or by an error on the standard I/O channels. In these cases an appropriate error code will be returned to MSX-DOS.

A transient program can define an "abort routine". This will be called to treat the abnormal termination of the program appropriately when the program terminates by a "Program Terminate" or "Terminate with error code" function, or after an abort error (see above). How to define this routine and for what may be used is described in the MSX-DOS Function Specification.

2.2.3 Page zero usage

On entry, various parameter areas are set up for the transient program in the first 256 bytes of RAM. The layout of this area is as below and is compatible with MSX-DOS 1 and with CP/M apart from the area used for MSX slot switching calls.[1]



[1]Figure

```
      .   .   original command line parameters.                    .
      .   .                                                        .
    00F8h |                                                        |
          +------+------+------+------+------+------+------+------+
```

At address `0000h` is a jump instruction which can be used for terminating the transient program. The destination of this jump can also be used to locate the BIOS jump vector (see section 2.2.4). The low byte of this jump address will always be `03h` for CP/M compatibility.

The two reserved bytes at addresses `0003h` and `0004h` are the `IOBYTE` and current drive/user in CP/M. Although MSX-DOS keeps the current drive byte up to date for CP/M compatibility, new programs are not recommended to use this but instead to use the "Get current drive" MSX-DOS function call. The user number and `IOBYTE` are not supported since I/O redirection is not done in the same way as CP/M and there is no concept of user numbers.

At address `0005h` is a jump instruction to the start of the resident part of MSX-DOS which is used for making MSX-DOS calls. In addition the address of this jump defines the top of the TPA which the program may use. The size of the TPA depends on what cartridges are used on the MSX machine and the number of them, but is typically 53K. The low byte of the destination of this jump will always be `06h` for CP/M compatibility, and the six bytes immediately preceding it will contain the CP/M version number and a serial number.

Four bytes are reserved for the user at each Z80 restart location (`0008h`-`0028h`), which is sufficient for a jump. The bytes between the restart locations however are used for the entry points to various MSX slot switching routines.

The whole area from `0038h` to `005Bh` is used for MSX interrupt and secondary slot switching code, and must not be modified. Note that most CP/M debuggers (such as ZSID and DDT) use address `38h` as a breakpoint entry, and these programs will have to be modified to use a different restart. `RST 28h` is recommended.

The two FCBs set up at addresses `005Ch` and `006Ch` are valid unopened FCBs containing the first two command line parameters interpreted as filenames. If both filenames are to be used then the second one must be copied to a separate FCB elsewhere in memory because it will be overwritten when the first one is opened. See section 2.3.6 for the format of FCBs.

The whole of the command line, with the initial command removed, is stored in the default disk transfer area at address `0080h`, with a length byte first and a terminating null (neither the null nor the length byte are included in the length). This string will have been upper-cased (when the environment string UPPER is ON) and will include any leading spaces typed to ensure CP/M compatibility.

New programs for MSX-DOS should not use the CP/M FCBs, since other MSX-DOS calls are available which are generally easier to use and which allow programs to access directories and handle path names (see section 2.3 for details of these facilities).

Improved methods are also available for accessing the command line. An environment string called `"PARAMETERS"` is set up which contains the command line

not upper-cased. Another environment string called "PROGRAM" allows programs to find out the drive, directory and filename from which they were loaded. See section 2.3.5 for details of these environment strings and of environment strings in general.

2.2.4 BIOS jump table

The jump at address 0000h will always jump to an address whose low byte is 03h. At this address will be another jump instruction which is the second entry in a seventeen entry jump table. This corresponds exactly to the BIOS jump table in CP/M 2.2.

The first eight entries in the table are for rebooting and for character I/O. These routines are implemented with the same specification as CP/M. The remaining jumps are low level disk related functions in CP/M and have no equivalent in MSX-DOS since its filing system is totally different. These routines simply return without doing anything apart from corrupting the main registers and returning an error where possible.

MSX-DOS switches to an internal stack while executing a BIOS call and so only a small amount of space (8 bytes) is required on the user's stack.

Note that although the jump table is always on a 256 byte page boundary, it is not the "correct" distance above the top of the TPA (as defined by the contents of address 0006h) to correspond with CP/M 2.2. This should not matter to well behaved CP/M programs but it is rumoured that some programs rely on the size of the BDOS in CP/M 2.2. These programs will need modification.

The entries in the BIOS jump vector are as below:

```
xx00h - JMP WBOOT  ;Warm boot
xx03h - JMP WBOOT  ;Warm boot
xx06h - JMP CONST  ;Console status
xx09h - JMP CONIN  ;Console input
xx0Ch - JMP CONOUT ;Console output
xx0Fh - JMP LIST   ;List output
xx12h - JMP PUNCH  ;Punch (auxiliary) output
xx15h - JMP READER ;Reader (auxiliary) input
xx18h - JMP RETURN ;Home in CP/M
xx1Bh - JMP RETURN ;Select disk in CP/M
xx1Eh - JMP RETURN ;Set track in CP/M
xx21h - JMP RETURN ;Set sector in CP/M
xx24h - JMP RETURN ;Set DMA address in CP/M
xx27h - JMP RETURN ;Read sector in CP/M
xx2Ah - JMP RETURN ;Write sector in CP/M
xx2Dh - JMP LSTST  ;List status
xx30h - JMP RETURN ;Sector translate in CP/M
```

2.2.5 RAM paging

When a transient program is loaded, the mapper RAM slot will be enabled in all four pages and the four RAM segments which make up the basic 64k will be paged in. There will be MSX BIOS ROM compatible slot handling entry points available in page-0 and various mapper support routines available in page-3 (see section 5 for specifications of these).

A program may do any slot switching and paging which it likes while it is running and need not restore either the slot selections or the RAM paging before it exits, since `COMMAND2.COM` will handle this. A program must of course take the usual precautions with the interrupt and the slot entry points if it alters page-0, and must never alter page-3 (nothing is allowed to do that!).

Pages 0, 1 and 2 can contain any slot when doing a function call and will be preserved. Any parameters can be passed from the slot being selected, except that environment strings and disk transfer areas must be in the mapper RAM slot.

Any RAM segments can be selected in pages 0, 1 and 2 when an MSX-DOS function call or an MSX-DOS BIOS function call is made, and also the stack can be in any page. The current paging state will be preserved by all function calls even in error conditions. Any disk transfers will be done to the RAM segments which are paged in when the function call is made, even if they are not the original TPA segments.

If a transient program wants to use more RAM than the TPA then it can use the mapper support routines (described in section 2.5) to obtain more RAM. Before using any RAM other than the four TPA segments, the program must ask the mapper routines to allocate a new segment. This ensures that there is no contention with the program trying to use a segment which is already in use (by the RAM disk for example). The segments should normally be allocated as "user segments" since these will automatically be freed when the program terminates. "system segments" should only be allocated if it is necessary for them to remain in use after the transient program has terminated.

Having allocated additional segments, the program may page them in and use any of the mapper support routines to access them. It will normally be necessary for a transient program to remember the segment numbers of the TPA segments in order to page them back in when they are required. The segment numbers will normally be 0, 1, 2 and 3 but this must NOT be assumed by transient programs, they must use the `"GET_Pn"` mapper routines to find out the segment numbers before paging anything else in.

2.3 MSX-DOS function calls

2.3.1 Calling conventions

MSX-DOS function calls are made by putting the function code in register C with any other parameters required in the other main registers and then

executing a "CALL 5" instruction. The operation will be performed and the results will be returned in various registers depending on the function.

Generally all the main registers (AF, BC, DE and HL) will be corrupted by MSX-DOS calls or will return results. The alternate register set (AF', BC', DE' and HL') are always preserved, whilst the index registers (IX and IY) are preserved except in the one or two cases where they return results.

Only a small amount of space (8 bytes) is needed on the transient program's stack because MSX-DOS switches to an internal stack when it is called.

For compatibility all functions which have a CP/M counterpart return with A=L and B=H. Frequently A returns an error flag, with zero indicating success and 01h or FFh indicating failure.

All of the new MSX-DOS functions (function code above 40h) return with an error code in A and any other results in the other main registers. An error code of 0 means no error, whilst any non-zero code means an error occurred, the exact nature of which can be found by looking at the value. A list of error codes and messages is given in section 2.6. An "explain" function is provided which will give an ASCII explanation string for an error code (see chapter 3 for details).

The actual functions available are documented in chapter 3.

2.3.2 Devices and character I/O

Wherever a filename is given to an MSX-DOS function, a device name may also be given. These devices are used for character based I/O and allow a program to access a disk file or a character device in exactly the same way without having to know which it is using.

The syntax of device names is identical to that of filenames so programs do not need any special handling to use device names. This applies both to the new MSX-DOS 2 functions and the CP/M compatible FCB functions. The reserved filenames used for devices are:

```
CON - screen output, keyboard input
PRN - printer output
LST - printer output
AUX - auxiliary output/input
NUL - null device
```

When any of these appear as the main part of a filename, it actually refers to the device; the extension is ignored. Most function calls that use files can also use devices. For example, a filename of CON can be successfully given to the "rename file" function or the "delete file" function. No error will be returned but the device will be unaffected.

The AUX device mentioned above does not do anything by default, but a routine may be hooked into it so that it refers for example to a serial driver. The NUL device does not actually do anything; output characters are ignored and an end-of-file is always input. The LST and PRN devices are identical.

The CON device is used to read from the keyboard or write to the screen. When reading from the CON device, the system will read a line at a time allowing the user to use line editing facilities. Only when the user types a CR (carriage return) will the line be entered. End of input is signified by a CTRL-Z character at the start of a line.

The system automatically opens several file handles to these standard devices (see section 2.3.3 for details). These file handles may be used by programs foraccessing the standard devices. Alternatively a program can do character I/O by using the traditional CP/M character functions (functions 01h...0Bh). These two methods are both acceptable but they should not normally be mixed since they use separate buffering schemes and so characters can get lost in these buffers.

The redirection is specified at the command line, both of these methods (standard file handles and character functions) will be redirected. However it is preferable to use the standard file handles and to read and write in large blocks because when accessing disk files these will be very much faster than using the character functions.

Even if the redirection was specified at the command line, programs may sometimes need to do screen output and keyboard input which will bypass any redirection. For example disk error handling routines may need to do this. To facilitate this, there is a function provided which allows redirection of the character functions to be temporarily cancelled. This is described in chapter 3 (function number 70h).

2.3.3 File handles

File handles are the method by which files are read from and written to using the new MSX-DOS functions. File handles may also be used to manipulate files in other ways (e.g. the manipulation of the file attributes).

A file handle is an 8 bit number which refers to a particular open file or device. A new file handle is allocated when the "open file handle" (function 43H) or "create file handle" (function 44H) function is used. The file handle can be used to read and write data to the file and remains in existence until the "close file handle" (function 45H) or "delete file handle" (function 52H) function is called. Other operations can be done on files using file handles, such as changing the attributes or renaming the files to which they refer.

Whenever MSX-DOS allocates a new file handle, it will use the lowest number available. The maximum file handle number in the current version is 63. In future versions this may be increased but will never be greater than 127, so file handles can never be negative.

Space for the internal data structures used for file handles is allocated dynamically within a 16K RAM segment (the "data segment") so there is no fixed limit to the number of file handles which can be open at one time. This segment is kept outside the TPA, so anything stored there does not reduce TPA size. As well as keeping internal file handle information, the system also keeps disk buffers and environment strings in the data segment.

Various file handles are pre-defined and are already open when a transient program is executed. These file handles refer to the standard input and output devices (see section 2.3.2). The "traditional" CP/M style MSX-DOS character I/O functions actually refer to these file handles.

A transient program actually gets a copy of the standard input and output file handles which the command interpreter was using, rather than the originals. This means that the program can freely close these file handles and re-open them to different destinations and need not reset them before terminating.

The default file handles and their destinations are:

```
0 - Standard input (CON)
1 - Standard output (CON)
2 - Standard error input/output (CON)
3 - Standard auxiliary input/output (AUX)
4 - Standard printer output (PRN)
```

When the command interpreter is about to execute a command (for example a transient program), it executes a "fork" function call (function `60H`). This informs the system that a new program is being executed as a "subroutine" and amongst other things, all of the currently open file handles are duplicated, so that the new program will be using copies of the original handles, rather than the command interpreter's.

If the transient program changes any of the file handles, by closing any existing ones or opening new ones, it will be the program's own set of file handles which are modified, the original set will remain unaltered. After the program has terminated, the command interpreter executes a "join" function call (function `61H`), passing to it a process id which was returned from the original "fork". This tells the system that the new program has terminated and so all its file handles can be thrown away.

Reference counts are kept of how many copies of each handle there are which enables the system to tidy up any file handles which are no longer needed when a program terminates. This ensures that the system will not run out of file handles because of badly behaved programs not closing them.

These "fork" and "join" functions are available for user programs if they find them useful. In addition to tidying up file handles, "join" will also free up any user allocated RAM segments which the program has not freed.

2.3.4 File info blocks

All new MSX-DOS functions that act on files on disk can be passed a simple pointer to a null-terminated string (called an ASCIIZ string), which can contain a drive, path and unambiguous filename. These are typically the operations which a transient program will perform, often through a high level language interface. The Command Specification gives details of these.

To any of these ASCIIZ functions, a File Info Block (FIB) may passed instead. FIBs are used for more complex operations such as the searching of directories for unknown files or sub-directories.

A FIB is a 64 byte area of the user's memory which contains information about the directory entry on disk for a particular file or sub-directory. The information in a FIB is filled in by the new MSX-DOS "find" functions ("find first entry" (function 40H), "find new entry" (function 42H) and "find next entry" (function 41H)). The format of a File Info Block is as follows:

```
0      - Always 0FFh
1..13  - Filename as an ASCIIZ string
14     - File attributes byte
15..16 - Time of last modification
17..18 - Date of last modification
19..20 - Start cluster
21..24 - File size
25     - Logical drive
26..63 - Internal information, must not be modified
```

The 0FFh at the start of the fileinfo block must be there to distinguish it from a pathname string, since some functions can take either type of parameter.

The filename is stored in a suitable format for directly printing, and is in the form of an ASCIIZ string. Any spaces will have been removed, the filename extension (if present) will be preceded by a dot and the name will have been uppercased. If the entry is a volume label then the name will be stored without the "." separator, with spaces left in and not uppercased.

The file attributes byte is a byte of flags concerning the file. The format of this byte is:

Bit 0 READ ONLY. If set then the file cannot be written to or deleted, but can be read, renamed or moved.

Bit 1 HIDDEN FILE. If set then the file will only be found by the "Find First" function if the "hidden file" bit is set in the search attributes byte. All the commands implemented by the command interpreter that access files and directories on disk can take a "/H" option which allows the command to find hidden files.

Bit 2 SYSTEM FILE. As far as MSX-DOS functions are concerned, this bit has exactly the same effect as the "HIDDEN FILE" bit except that the "Find New" and "Create" function calls will not automatically delete a system file. None of the commands implemented by the command interpreter allow system files to be accessed.

Bit 3 VOLUME NAME. If set then this entry defines the name of the volume. Can only occur in the root directory, and only once. All other bits are ignored.

Bit 4 DIRECTORY. If set then the entry is a sub-directory rather than a file and so cannot be opened for reading and writing. Only the hidden bit has any meaning for sub-directories.

Bit 5 ARCHIVE BIT. Whenever a file has been written to and closed this bit is set. This bit can be examined by, for example, the `XCOPY` command to determine whether the file has been changed.

Bit 6 Reserved (always 0).

Bit 7 DEVICE BIT. This is set to indicate that the FIB refers to a character device (eg. `"CON"`) rather than a disk file. All of the other attributes bits are ignored.

The time of last modification is encoded into two bytes as follows:

```
Bits 15..11 - HOURS (0..23)
Bits 10...5 - MINUTES (0..59)
Bits  4...0 - SECONDS/2 (0..29)
```

The date of last modification is encoded into two bytes as follows. If all bits are zero then there is no valid date set.

```
Bits 15...9 - YEAR (0..99 corresponding to 1980..2079)
Bits  8...5 - MONTH (1..12 corresponding to Jan..Dec)
Bits  4...0 - DAY (1..31)
```

The file size is a 32 bit number stored with the lowest byte first, and is zero for sub-directories.

The logical drive is a single byte drive number, with 1 corresponding to `A:`, 2 to `B:` etc. It will never be zero, since if zero was specified in the original function, this means the default drive and the driven number of the default drive will be filled in here.

The internal information tells MSX-DOS where on the disk the directory entry is stored. This enables functions to which the fileinfo block is passed to operate on the directory entry, for example deleting it, renaming it or opening it. Data stored here also enables the "find next entry" function (function `41H`) to carry on the search to find the next matching file. The user should not access or modify the internal information at all.

Fileinfo blocks are filled in by the "find first entry", "find new entry" and "find next entry" MSX-DOS functions. Each of these functions locates a directory entry and fills in the fileinfo block with the relevant information.

In the case of "find first entry" a directory will be searched for the first entry which matches a given filename and which has suitable attributes (see the Function Specification for details). "Find next entry" carries on a search started by a previous "find first entry" function and updates the fileinfo block with the next matching entry.

"Find new entry" is just like "find first entry" except that instead of looking for a matching entry, it will create a new one and then return a fileinfo block just as if "find first" had found it.

Having created a fileinfo block using one of the "find" functions there are two ways in which it can be used. The first way is to simply use the information

which it contains such as the filename and size. For example the "DIR" command simply prints out the information on the screen.

The more interesting way of using a fileinfo block is to pass it back to another MSX-DOS function in order to carry out some operation on the directory entry. Many of the MSX-DOS functions described in the Function Specification take a pointer in register DE which can either point to a drive/path/file string or a fileinfo block. In either case a particular file or directory is being specified for the function to act on.

The functions which can take such a parameter are "Delete File or Subdirectory" (function 4DH), "Rename file or Subdirectory" (function 4EH), "Move File or Subdirectory" (function 4FH), "Get/Set File Attributes" (function 50H), "Get/Set File Date and Time" (function 51H) and "Open File handle" (function 43H). All of these carry out the obvious function on the specified file or directory.

A fileinfo block can also be passed to a "find first" or "find new" function in place of the drive/path/file string. In this case the fileinfo block must refer to a directory rather than a file and a filename string must also be passed in register HL (typically null which is equivalent to "*.*"). The directory specified by the fileinfo block will be searched for matches with the filename, subject to the usual attribute checking. This feature is necessary for the command interpreter so that a command such as "DIR A:UTIL" can have the required action if UTIL is a directory.

2.3.5 Environment strings

MSX-DOS maintains a list of "environment strings" in it's data segment. An environment string is a named item which has a value associated with it. Both the name and the value are user-defined. Environment strings are accessed at the function call level via the "Get Environment String" (function 6BH), "Set Environment String" (function 6CH) and "Find Environment String" (function 6DH) functions.

The name of an environment string is a non-null string consisting of any characters that are valid in filenames. The name can be up to 255 characters long. The characters of the name are upper-cased when the string is defined, although for name comparisons case is not significant.

The value of an environment string consists of a string of non-null characters and can be up to 255 characters long. If the value of an environment string is set to a null string, then the name is removed from the list of environment strings. Similarly, if the value of an environment string that has not been defined is read, then a null string is returned. The value is not upper-cased and no interpretation is placed on the characters in the value string.

When a transient program is loaded and executed from COMMAND2.COM, two special environment strings are set up, which it can read.

An environment string called PARAMETERS is set up to the command line not including the actual command name. This is similar to the one set up at 80h for CP/M compatibility, but has not been upper-cased.

Another environment string called PROGRAM is also set up and this is the whole path used to locate the program on disk. The drive is first, followed by the path from the root and then the actual filename of the program. The drive, path and filename can be separated if desired using the "Parse Pathname" function call (function 5CH).

The PROGRAM environment string has several uses. The main use is that a program can use it to load overlay files from the same directory as it was loaded from. The last item in PROGRAM (ie. the actual program filename) is replaced with the name of the overlay file, and then the new string can be passed to any of the new MSX-DOS 2 functions that take ASCIIZ strings (such as "Open File").

Note that some CP/M programs are capable of loading and running transient programs, and in this case they obviously will not have set up the PROGRAM and PARAMETERS environment strings, and they will in fact still be set up from when the CP/M program was loaded. If a program wishes to use PROGRAM and PARAMETERS and still be loadable from a CP/M program, then it can look at a variable called "LOAD_FLAG", which is in page 0 at address 0037h. It is set to zero on every MSX-DOS 2 function call but is set to non-zero immediately before a transient program is executed by the command interpreter. Similarly, if a new transient program has the ability to load other transient programs and it sets up PROGRAM and PARAMETERS, then it should also set LOAD_FLAG to non-zero.

Another special environment string is one called APPEND. This can be set up by the user from the command interpreter and is used by the CP/M "Open File (FCB)" function. When this function call is done and the file is not found, an alternative directory, specified by APPEND, is searched. It is not anticipated however that new transient programs will use this function call or the APPEND environment string.

Several environment strings are set up by the command interpreter when it starts up and are altered by the user to control various system features and options, and it may be useful for transient programs to read some of these. For example, it may be useful to read the PATH environment string or the DATE and TIME environment strings if the program prints out dates and times. The Command Specification contains details of these default environment strings.

2.3.6 File control blocks

It is not anticipated that specially written MSX-DOS 2 transient programs or MSX-DOS 1 or CP/M programs which are modified for MSX-DOS 2 will use the CP/M-compatible FCB functions, but the format of the FCBs used for these functions is given here for reference. This format is, of course, very similar to the FCBs used by CP/M and MSX-DOS 1 but the use of some of the fields within the FCB are different (though generally compatible).

A basic FCB is 33 bytes long. This type of FCB can be used for file management operations (delete, rename etc.) and also for sequential reading and writing. The random read and write functions use an extra 3 bytes on the end

of the FCB to store a random record number. The MSX-DOS 1 compatible block read and write functions also use this additional three (or in some cases four) bytes - see chapter 3 for details.

The layout of an FCB is given below. A general description of each of the fields is included here. The individual function description given in the Function Specification details of how the fields are used for each function where this is not obvious.

`00h` Drive number 1...8. 0 => default drive. Must be set up in all FCBs used, never modified by MSX-DOS function calls (except "Open File" if `APPEND` was used).

`01h...08h` Filename, left justified with trailing blanks. Can contain "?" characters if ambiguous filename is allowed (see chapter 3). When doing comparisons case will be ignored. When creating new files, name will be uppercased.

`09h...0Bh` Filename extension. Identical to filename. Note that bit-7 of the filename extension characters are NOT interpreted as flags as they are in CP/M.

`0Ch` Extent number (low byte). Must be set (usually to zero) by the transient program before open or create. It is used and updated by sequential read and write, and also set by random read and write. This is compatible with CP/M and MSX-DOS 1.

`0Dh` File attributes. Set by "open", "create" or "find".

`0Eh` Extent number (high byte) for CP/M functions. Zeroed by open and create. For sequential read and write it is used and updated as an extension to the extent number to allow larger files to be accessed. Although this is different from CP/M it does not interfere with CP/Ms use of FCBs and is the same as MSX-DOS 1.

Record size (low byte) for MSX-DOS 1 compatible block functions. Must be set to the required record size before using the block read or write functions.

`0Fh` Record count for CP/M functions. Set up by open and create and modified when necessary by sequential and random reads and writes. This is the same as CP/M and MSX-DOS 1.

Record size (high byte) for MSX-DOS 1 compatible block functions. Must be set to the required record size before using the block read and write functions.

`10h...13h` File size in bytes, lowest byte first. File size is exact, not rounded up to 128 bytes. This field is set up by open and create and updated when the file is extended by write operations. Should not be

modified by the transient program as it is written back to disk by a close function call. This is the same as MSX-DOS 1 but different from CP/M which stores allocation information here.

`14h...17h` Volume-id. This is a four byte number identifying the particular disk which this FCB is accessing. It is set up by open and create and is checked on read, write and close calls. Should not be modified by the program. Note that this is different from MSX-DOS 1 which stores the date and time of last update here, and from CP/M which stores allocation information.

`18h...1Fh` Internal information. These bytes contain information to enable the file to be located on the disk. Should not be modified at all by the transient program. The internal information kept here is similar but not identical to that kept by MSX-DOS 1 and totally different from CP/M.

`20h` Current record within extent (0...127). Must be set (normally to zero) by the transient program before first sequential read or write. Used and modified by sequential read and write. Also set up by random read and write. This is compatible with CP/M and MSX-DOS 1.

`21h...24h` Random record number, low byte first. This field is optional, it is only required if random or block reads or writes are used. It must be set up before doing these operations and is updated by block read and write but not by random read or write. Also set up by the "set random record" function.

For the block operations, which are in MSX-DOS 1 but not in CP/M, all four bytes are used if the record size is less than 64 bytes, and only the first three bytes are used if the record size is 64 bytes or more. For random read and write only the first three bytes are used (implied record size is 128 bytes). This is compatible with CP/M and with MSX-DOS 1.

2.4 Screen control codes

Below is a list of all control codes and escape sequences which may be used when doing character output by MSX-DOS character functions, BIOS calls or writing to the device `CON`. These are all compatible with MSX-DOS 1 and contain the VT-52 control codes.

The screen is 24 lines of 2 to 80 characters. When a printing character is displayed the cursor is moved to the next position and to the start of the next line at the end of a line. If a character is written in the bottom right position then the screen will be scrolled to allow the cursor to be positioned at the start of the next line. The letters in escape sequences must be in the correct case, the

spaces are inserted for readability they are not part of the sequence. Numbers (indicated by <n> or <m>) are included in the sequence as a single byte usually with an offset of 20h added.

CTRL-G 07h = Bell

CTRL-H 08h = Cursor left, wraps around to previous line, stop at top left of screen.

CTRL-I 09h = Tab, overwrites with spaces up to next 8th column, wraps around to start of next line, scrolls at bottom right of screen.

CTRL-J 0Ah = Line feed, scrolls if at bottom of screen.

CTRL-K 0Bh = Cursor home.

CTRL-L 0Ch = Clear screen and home cursor.

CTRL-M 0Dh = Carriage return.

CTRL-[1Bh = ESC - see below for escape sequences.

CTRL-\ 1Ch = Cursor right, wrap around to next line, stop at bottom right of screen.

CTRL-] 1Dh = Cursor left, wrap around to previous line, stop at top left of screen.

CTRL-^ 1Eh = Cursor up, stop at top of screen.

CTRL-_ 1Fh = Cursor down, stop at bottom of screen.

7Fh = Delete character and move cursor left, wrap around to previous line, stop at top of screen.

ESC-A Cursor up, stops at top of screen.

ESC-B Cursor down, stops at bottom of screen.

ESC-C Cursor right, stops at end of line.

ESC-D Cursor left, stops at start of line.

ESC-E Clear screen and home cursor.

ESC-H Cursor home.

ESC-J Erase to end of screen, don't move cursor.

ESC-j Clear screen and home cursor.

ESC-K Erase to end of line, don't move cursor.

`ESC-L` Insert a line above cursor line, scroll rest of screen down. Leave cursor at start of new blank line.

`ESC-l` Erase entire line, don't move cursor.

`ESC-M` Delete cursor line, scrolling rest of screen up. Leave cursor at start of next line.

`ESC x 4` Select block cursor.

`ESC x 5` Cursor off.

`ESC Y <n><m>` Position cursor at row <n> column <m>. Top left of screen is n=m=20h (ASCII space).

`ESC y 4` Select underscore cursor.

`ESC y 5` Cursor on.

2.5 Mapper support routines

MSX-DOS 2 contains routines to provide support for the memory mapper. This allows MSX application programs or MSX-DOS transient programs to utilize more than the basic 64k of memory, without conflicting with the RAM disk or any other system software.

2.5.1 Mapper initialization

When the DOS kernel is initialized it checks that there is the memory mapper in the system, and that there is at least 128k of RAM available. If the kernel has found at least one slot which contains 128k of the mapper RAM, it selects the slot which contains the largest amount of RAM (or the slot with the smallest slot number, if there are two or more mapper slots which have the same amount of RAM) and makes that slot usable as the system RAM. When there is not enough memory on the memory mapper, MSX-DOS 2 won't start.

Next the kernel builds up a table of all the 16k RAM segments available to this slot (primary mapper slot). The first four segments (64k) for the user and the two highest numbered segments are allocated to the system, one for the DOS kernel code and one for the DOS kernel workspace. All other segments (at least two) are marked as free initially. Then the kernel builds up the similar tables for other RAM slots, if any. All of these segments are marked as free initially.

2.5.2 Mapper variables and routines

The mapper support routines use some variables in the DOS system area. These tables may be referred and used by the user programs for the various purposes,

but must not be altered. The contents of the tables are as follows:

Address	Function
+0	Slot address of the mapper slot.
+1	Total number of 16k RAM segments. 1...255 (8...255 for the primary)
+2	Number of free 16k RAM segments.
+3	Number of 16k RAM segments allocated to the system (at least 6 for the primay)
+4	Number of 16k RAM segments allocated to the user.
+5..+7	Unused. Always zero.
+8...	Entries for other mapper slots. If there is none, +8 will be zero.

A program uses the mapper support code by calling various subroutines. These are accessed through a jump table which is located in the MSX-DOS system area. The contents of the jump table are as follows:

Address	Entry name	Function
+0H	ALL_SEG	Allocate a 16k segment.
+3H	FRE_SEG	Free a 16k segment.
+6H	RD_SEG	Read byte from address A:HL to A.
+9H	WR_SEG	Write byte from E to address A:HL.
+CH	CAL_SEG	Inter-segment call. Address in IYh:IX.
+FH	CALLS	Inter-segment call. Address in line after the call instruction.
+12H	PUT_PH	Put segment into page (HL).
+15H	GET_PH	Get current segment for page (HL).
+18H	PUT_P0	Put segment into page 0.
+1BH	GET_P0	Get current segment for page 0.
+1EH	PUT_P1	Put segment into page 1.
+21H	GET_P1	Get current segment for page 1.
+24H	PUT_P2	Put segment into page 2.
+27H	GET_P2	Get current segment for page 2.
+2AH	PUT_P3	Not supported since page-3 must never be changed. Acts like a "NOP" if called.
+2DH	GET_P3	Get current segment for page 3.

A program can use the extended BIOS calls for the mapper support to obtain these addresses. The calls are provided because these addresses may be changed in the future version, or to use mapper routines other than MSX-DOS mapper support routines.

To use the extended BIOS, the program should test "HOKVLD" flag at FB20h in page-3. If bit-0 (LSB) is 0, there is no extended BIOS nor the mapper support. Otherwise, "EXTBIO" entry (see below) has been set up and it can be called with various parameters. Note that this test is unnecessary for the applications which

are based on MSX-DOS (such as the program which is loaded from the disk), and the program may proceed to the next step.

Next, the program sets the device number of the extended BIOS in register D, the function number in register E, and required parameters in other registers, and then calls "EXTBIO" at FFCAh in page-3. In this case, the stack pointer must be in page-3. If there is the extended BIOS for the specified device number, the contents of the registers AF, BC and HL are modified according to the function; otherwise, they are preserved. Register DE is always preserved. Note that in any cases the contents of the alternative registers (AF', BC', DE' and HL') and the index registers (IX and IY) are corrupted.

The functions available in the mapper support extended BIOS are:

- Get mapper variable table

  Parameter: A = 0
  D = 4 (device number of mapper support)
  E = 1

  Result: A = slot address of primary mapper
  DE = reserved
  HL = start address of mapper variable table

- Get mapper support routine address

  Parameter: A = 0
  D = 4
  E = 2

  Result: A = total number of memory mapper segments
  B = slot number of primary mapper
  C = number of free segments of primary mapper
  DE = reserved
  HL = start address of jump table

In these mapper support extended BIOS, register A is not required to be zero. Note that, however, if there is no mapper support routine, the contents of registers will not be modified, and the value which is not zero will be returned in A otherwise. Thus, the existence of the mapper support routine can be determined by setting zero in A at the calling and examining the returned value of A.

The slot address of the primary mapper returned by the extended BIOS is the same as the current RAM slot address in page-3, and, in the ordinary environment (DISK-BASIC and MSX-DOS), the same RAM slot is also selected in page-2. In MSX-DOS, this is also true in page-0 and page-1.

2.5.3 Using mapper routines

A program can request a 16k RAM segment at any time by calling the "ALL_SEG" routine. This either returns an error if there are no free segments, or the segment number of a new segment which the program can use. A program must

not use any segment which it has not explicitly allocated, except for the four segments which make up the basic 64k of RAM.

A segment can be allocated either as a user segment or as a system segment. User segments will be automatically freed when the program terminates, whereas system segments are never freed unless the program frees them explicitly. Normally, programs should allocate user segments.

RAM segments can be accessed by the "RD_SEG" and "WR_SEG" routines which read and write bytes to specified segments. The routines "CAL_SEG" and "CALLS" allow inter-segment calls to be done in much the same way as inter-slot calls in the current MSX system.

Routines are provided to explicitly page a segment in, or to find out which segment is in a particular page. There are routines in which the page (0...3) is specified by the top two bits of an address in HL ("PUT_PH" and "GET_PH"). And there are also specific routines for accessing each page ("GET_Pn" and "PUT_Pn"). These routines are very fast so a program should not suffer in performance by using them.

Note that page-3 should never be altered since this contains the mapper support routines and all the other system variables. Also great care must be taken if page-0 is altered since this contains the interrupt and the slot switching entry points. Pages 1 and 2 can be altered in any way.

None of the mapper support routines will disturb the slot selection mechanism at all. For example when "PUT_P1" is called, the specified RAM segment will only appear at address 4000h...7FFFh if the mapper slot is selected in page-1. The "RD_SEG" and "WR_SEG" routines will always access the RAM segment regardless of the current slot selection in the specified page, but the mapper RAM slot must be selected in page-2.

2.5.4 Allocating and freeing segments

The following two routines can be called to allocate or free segments. All registers apart from AF and BC are preserved. An error is indicated by the carry flag being set on return. The slot selection and RAM paging may be in any state when these routines are called and both will be preserved. The stack must not be in page-0 or page-2 when either of these routines are called.

A program must not use any segment (apart from the four which make up the basic 64k) unless it has specifically allocated it, and must not continue to use a segment after it has been freed.

A segment may be allocated either as a user or a system segment. The only difference is that user segments will be automatically freed when the program terminates whereas system segments will not be. In general a program should allocate a user segment unless it needs the data in the segment to outlast the program itself. User segments are always allocated from the lowest numbered free segment and system segments from the highest numbered one.

An error from "allocate segment" usually indicates that there are no free segments, although it can also mean that an invalid parameter was passed in

register A and B. An error from "free segment" indicates that the specified segment number does not exist or is already free.

ALL_SEG

- Parameters

A=0	Allocate user segment
A=1	Allocate system segment
B=0	Allocate primary mapper
B!=0	Allocate FxxxSSPP slot address (primary mapper, if 0)
xxx=000	Allocate specified slot only
xxx=001	Allocate other slots than specified
xxx=010	Try to allocate specified slot and, if it failed, try another slot (if any)
xxx=011	Try to allocate other slots than specified and, if it failed, try specified slot

- Results

CARRY set	No free segments
CARRY clear	Segment allocated
A	New segment number
B	Slot address of mapper slot (0 if called as B=0)

FRE_SEG

- Parameters

A	Segment number to free
B=0	Primary mapper
B!=0	Mapper other than primary

- Results

CARRY set	Error
CARRY clear	Segment freed OK

2.5.5 Inter-segment read and write

The following two routines can be called to read or write a single byte from any mapper RAM segment. The calling sequence is very similar to the inter-slot read and write routines provided by the MSX system ROM. All registers apart from `AF` are preserved and no checking is done to ensure that the segment number is valid.

The top two bits of the address are ignored and the data will be always read or written via page-2, since the segment number specifies a 16k segment which could appear in any of the four pages. The data will be read or written from the correct segment regardless of the current paging or slot selection in page-0 or page-1, but note that the mapper RAM slot must be selected in page-2 when either of these routines are called. This is so that the routines do not have to do any slot switching and so can be fast. Also the stack must not be in page-2. These routines will return disabling interrupts.

`RD_SEG`

- Parameters

`A`	Segment number to read from
`HL`	Address within this segment

- Results

`A`	Value of byte at that address

  All other registers preserved

`WR_SEG`

- Parameters

`A`	Segment number to write to
`HL`	Address within this segment
`E`	Value to write

- Results

`A`	Corrupted

  All other registers preserved

2.5.6 Inter-segment calls

Two routines are provided for doing inter-segment calls. These are modelled very closely on the two inter-slot call routines provided by the MSX system ROM, and the specification of their usage is very similar.

No check is done that the called segment actually exists so it is the user's responsibility to ensure this. The called segment will be paged into the specified address page, but it is the user's responsibility to ensure that the mapper slot is enabled in this page, since neither of these routines will alter the slot selection at all. This is to ensure that they can be fast.

The routine cannot be used to do an inter-segment call into page-3. If this is attempted then the specified address in page-3 will simply be called without any paging, since page-3 must never be altered. Calling into page-0 must be done with some care because of the interrupt and other entry point. Also care must be taken that the stack is not paged out by these calls.

These routines, unlike inter-slot calls, do not disable interrupts before passing control to the called routine. So they return to the caller in the same state as before, unless the interrupt flag was modified by the called routine.

Parameters cannot be passed in registers `IX`, `IY`, `AF'`, `BC'`, `DE'` or `HL'` since these are used internally in the routine. These registers will be corrupted by the inter-segment call and may also be corrupted by the called routine. All other registers (`AF`, `BC`, `DE` and `HL`) will be passed intact to the called routine and returned from it to the caller.

`CAL_SEG`

- Parameters

`IY`	Segment number to be called
`IX`	Address to call

  `AF`, `BC`, `DE`, `HL` passed to called routine

  Other registers corrupted

- Results

  `AF`, `BC`, `DE`, `HL`, `IX` and `IY` returned from called routine.

  All others corrupted.

`CALLS`

- Parameters

  `AF`, `BC`, `DE`, `HL` passed to called routine..

  Other registers corrupted.

  Calling sequence:

```
CALL CALLS
DB   SEGMENT
DW   ADDRESS
```

- Results

  AF, BC, DE, HL, IX and IY returned from called routine.

  All others corrupted.

2.5.7 Direct paging routines

The following routines are provided to allow programs to directly manipulate the current paging state without having to access the hardware. Using these routines ensures compatibility with any changes to the details of the hardware. The routines are very fast and so using them will not compromise the performance of programs.

Routines are provided to directly read or write to any of the four page registers. No checking of the validity of the segment number is done so this is the user's responsibility. Note that the value written in the register is also written in memory and, if the register value is requested, the value stored in memory will be returned and the one in the register will never be read directly. This is done to avoid errors from hardware conflicts when there are two or more mapper registers in the system. The user should always manipulate the memory mapper through these routines.

The "GET" routines return values from internal images of the registers without actually reading the registers themselves. This ensures that if a segment is enabled by, for example, "PUT_P1" then a subsequent "GET_P1" call will return the actual value. Reading the mapper register may produce a different value because the top bits of the segment numbers are generally not recorded.

Although a "PUT_P3" routine is provided, it is in fact a dummy routine and will not alter the page-3 register. This is because the contents of the page-3 register should never be altered. The "GET_P3" routine does behave as expected to allow the user to determine what segment is in page-3.

Another pair of routines ("GET_PH" and "PUT_PH") is provided which are identical in function except that the page is specified by the top two bits of register H. This is useful when register HL contains an address, and these routines do not corrupt register HL. "PUT_PH" will never alter the page-3 register.

PUT_Pn

- Parameters

n	0,1,2 or 3 to select page
A	Segment number

- Results

  None. All registers preserved

`GET_Pn`

- Parameters

  `n` 0,1,2 or 3 to select page

- Results

  `A` Segment number

  All other registers preserved

`PUT_PH`

- Parameters

  `H` High byte of address
  `A` Segment number

- Results

  None. All registers preserved

`GET_PH`

- Parameters

  `H` High byte of address

- Results

  `A` Segment number

  All other registers preserved

Before using these direct paging routines to alter the paging state, a program should first use the `"GET_Pn"` routines to determine the initial four segments for when it needs to restore these. No program should assume fixed values for these initial segments since they are likely to change in future versions of the system.

2.6 Errors

All the new MSX-DOS 2 functions (function codes above `40h`) return an "error code" in `A`. This is zero if the operation was successful. If non-zero, then the error code explains the exact nature of the error.

Since MSX-DOS 2 performs an `"OR A"` instruction immediately before returning from a function call, a `"JR NZ"` instruction is often used in the transient program immediately after the `"CALL 5"` instruction to test whether an error occurred. Frequently the destination of this error jump just loads the error code

into B and does a "Terminate with Error Code" function. This then passes the error code back to the command interpreter which prints the appropriate message.

A transient program may also itself get the actual message for any error returned by an MSX-DOS 2 function call by using the "Explain Error Code" function. See the chapter 3 for details.

The error codes start at 0FFh and descend in value. Values less than 40h are *user errors* and will never be used by the system and can be used by transient programs to return their own errors. User errors below 20h returned to the command interpreter will not have any message printed.

If the "Explain Error Code" function call (see chapter 3) is asked to explain an error code for which it does not have a message, then the string returned will be "System error <n>" or "User error <n>" as appropriate, where <n> is the error number.

Below is a list of all currently defined error numbers and their messages and meanings. Also given is the mnemonic, which is often used as a symbol in a source file and is used throughout the MSX-DOS 2 system to refer to a particular error.

2.6.1 Disk errors

The errors in this group are those which are usually passed to disk error handling routines. By default they will be reported as "Abort, Retry" errors. These errors except the one from "format disk" will be passed to the error handling routine, so they will not be returned as the return value from BDOS.

`Incompatible disk (.NCOMP, 0FFh)`

The disk cannot be accessed in that drive (eg. a double sided disk in a single sided drive).

`Write error (.WRERR, 0FEh)`

General error occurred during a disk write.

`Disk error (.DISK, 0FDh)`

General unknown disk error occurred.

`Not ready (.NRDY, 0FCh)`

Disk drive did not respond, usually means there is no disk in the drive.

`Verify error (.VERFY, 0FBh)`

With VERIFY enabled, a sector could not be read correctly after being written.

`Data error (.DATA, 0FAh)`

A disk sector could not be read because the CRC error checking was incorrect, usually indicating a damaged disk.

`Sector not found (.RNF, 0F9h)`

The required sector could not be found on the disk, usually means a damaged disk.

`Write protected disk (.WPROT, 0F8h)`

Attempt to write to a disk with the write protect tab on.

`Unformatted disk (.UFORM, 0F7h)`

The disk has not been formatted, or it is a disk using a different recording technique.

`Not a DOS disk (.NDOS, 0F6h)`

The disk is formatted for another operating system and cannot be accessed by MSX-DOS.

`Wrong disk (.WDISK, 0F5h)`

The disk has been changed while MSX-DOS was accessing it. Must replace the correct disk.

`Wrong disk for file (.WFILE, 0F4h)`

The disk has been changed while there is an open file on it. Must replace the correct disk.

`Seek error (.SEEK, 0F3h)`

The required track of the disk could not be found.

`Bad file allocation table (.IFAT, 0F2h)`

The file allocation table on the disk has got corrupted. `CHKDSK` may be able to recover some of the data on the disk.

`(.NOUPB, 0F1h)`

This error has no message because it is always trapped internally in MSX-DOS as part of the disk change handling.

`Cannot format this drive (.IFORM, 0F0h)`

Attempt to format a drive which does not allow formatting. Usually as a result of trying to format the RAM disk.

2.6.2 MSX-DOS function errors

The following errors are those which are normally returned from MSX-DOS function calls. See the chapter 3 for details of errors from particular MSX-DOS functions.

`Internal error (.INTER, 0DFh)`

Should never occur.

`Not enough memory (.NORAM, 0DEh)`

MSX-DOS has run out of memory in its 16k kernel data segment. Try reducing the number of sector buffers or removing some environment strings. Also occurs if there are no free segments for creating the RAMdisk.

`Invalid MSX-DOS call (.IBDOS, 0DCh)`

An MSX-DOS call was made with an illegal function number. Most illegal function calls return no error, but this error may be returned if a `"get previous error code"` function call is made.

`Invalid drive (.IDRV, 0DBh)`

A drive number parameter, or a drive letter in a drive/path/file string is one which does not exist in the current system.

`Invalid filename (.IFNM, 0DAh)`

A filename string is illegal. This is only generated for pure filename strings, not drive/path/file strings.

`Invalid pathname (.IPATH, 0D9h)`

Can be returned by any function call which is given an ASCIIZ drive/path/file string. Indicates that the syntax of the string is incorrect in some way.

`Pathname too long (.PLONG, 0D8h)`

Can be returned by any function call which is given an ASCIIZ drive/path/file string. Indicates that the complete path being specified (including current directory if used) is longer than 63 characters.

`File not found (.NOFIL, 0D7h)`

Can be returned by any function which looks for files on a disk if it does not find one. This error is also returned if a directory was specified but not found. In other cases, `.NODIR` error (see below) will be returned.

`Directory not found (.NODIR, 0D6h)`

Returned if a directory item in a drive/path/file string could not be found.

`Root directory full (.DRFUL, 0D5h)`

Returned by `"create"` or `"move"` if a new entry is required in the root directory and it is already full. The root directory cannot be extended.

`Disk full (.DKFUL, 0D4h)`

Usually results from a write operation if there was insufficient room on the disk for the amount of data being written. May also result from trying to create or extend a sub-directory if the disk is completely full.

`Duplicate filename (.DUPF, 0D3h)`

Results from `"rename"` or `"move"` if the destination filename already exists in the destination directory.

`Invalid directory move (.DIRE, 0D2h)`

Results from an attempt to move a sub-directory into one of its own descendants. This is not allowed as it would create an isolated loop in the directory structure.

`Read only file (.FILRO, 0D1h)`

Attempt to write to or delete a file which has the `"read only"` attribute bit set.

`Directory not empty (.DIRNE, 0D0h)`

Attempt to delete a sub-directory which is not empty.

`Invalid attributes (.IATTR, 0CFh)`

Can result from an attempt to change a file's attributes in an illegal way, or trying to do an operation on a file which is only possible on a sub-directory. Also results from illegal use of volume name fileinfo blocks.

`Invalid . or .. operation (.DOT, 0CEh)`

Attempt to do an illegal operation on the "." or ".." entries in a sub-directory, such as rename or move them.

`System file exists (.SYSX, 0CDh)`

Attempt to create a file or sub-directory of the same name as an existing system file. System files are not automatically deleted.

`Directory exists (.DIRX, 0CCh)`

Attempt to create a file or sub-directory of the same name as an existing sub-directory. Sub-directories are not automatically deleted.

`File exists (.FILEX, 0CBh)`

Attempt to create a sub-directory of the same name as an existing file. Files are not automatically deleted when creating sub-directories.

`File already in use (.FOPEN, 0CAh)`

Attempt to delete, rename, move, or change the attributes or date and time of a file which has a file handle already open to it, other than by using the file handle itself.

`Cannot transfer above 64K (.OV64K, 0C9h)`

Disk transfer area would have extended above `0FFFFh`.

`File allocation error (.FILE, 0C8h)`

The cluster chain for a file was corrupt. Use `CHKDSK` to recover as much of the file as possible.

`End of file (.EOF, 0C7h)`

Attempt to read from a file when the file pointer is already at or beyond the end of file.

`File access violation (.ACCV, 0C6h)`

Attempt to read or write to a file handle which was opened with the appropriate access bit set. Some of the standard file handles are opened in read only or write only mode.

`Invalid process id (.IPROC, 0C5h)`

Process id number passed to "`join`" function is invalid.

`No spare file handles (.NHAND, 0C4h)`

Attempt to open or create a file handle when all file handles are already in use. There are 64 file handles available in the current version.

`Invalid file handle (.IHAND, 0C3h)`

The specified file handle is greater than the maximum allowed file handle number.

`File handle not open (.NOPEN, 0C2h)`

The specified file handle is not currently open.

`Invalid device operation (.IDEV, 0C1h)`

Attempt to use a device file handle or fileinfo block for an invalid operation such as searching in it or moving it.

`Invalid environment string (.IENV, 0C0h)`

Environment item name string contains an invalid character.

`Environment string too long (.ELONG, 0BFh)`

Environment item name or value string is either longer than the maximum allowed length of 255, or is too long for the user's buffer.

`Invalid date (.IDATE, 0BEh)`

Date parameters passed to "`set date`" are invalid.

`Invalid time (.ITIME, 0BDh)`

Time parameters passed to "`set time`" are invalid.

`RAM disk (drive H:) already exists (.RAMDX, 0BCh)`

Returned from the "`ramdisk`" function if trying to create a RAM disk when one already exists.

`RAM disk does not exist (.NRAMD, 0BBh)`

Attempt to delete the RAM disk when it does not currently exist. A function which tries to access a non-existent RAM disk will get a `.IDRV` error.

`File handle has been deleted (.HDEAD, 0BAh)`

The file associate with a file handle has been deleted so the file handle can no longer be used.

`(.EOL, 0B9h)`

Internal error. Should never occur.

`Invalid sub-function number (.ISBFN, 0B8h)`

The sub-function number passed to the `IOCTL` function (function `4Bh`) was invalid.

2.6.3 Program termination errors

The following errors are those which may be generated internally in the system and passed to `"abort"` routines. They will not normally be returned from function calls. Note that an abort routine can also be passed any error which a transient program passes to the `"terminate with error code"` function call.

`Ctrl-STOP pressed (.STOP, 09Fh)`

The `CTRL-STOP` key is tested in almost all places in the system including all character I/O.

`Ctrl-C pressed (.CTRLC, 09Eh)`

`CTRL-C` is only tested for on those character functions which specify status checks.

`Disk operation aborted (.ABORT, 09Dh)`

This error occurs when any disk error is aborted by the user or automatically by the system. The original disk error code will be passed to the abort routine in `B` as the secondary error code.

`Error on standard output (.OUTERR, 09Ch)`

Returned if any error occurred on a standard output channel while it was being accessed through the character functions (functions `01h...0Bh`). The original error code is passed to the abort routine in register `B` as the secondary error code. This error will normally only occur if a program has altered the standard file handles.

`Error on standard input (.INERR, 09Bh)`

Returned if any error occurred on a standard input channel while it was being accessed through the character functions (functions `01h...0Bh`). The original error code is passed to the abort routine in register `B` as the secondary error code. The most likely error is end of file (`.EOF`). This error will normally only occur if a program has altered the standard file handles.

2.6.4 Command errors

The following errors will not be returned from an MSX-DOS function call, but are used by the command interpreter. They are included here because a transient program may find it useful to return some of them. Chapter 1 gives more details of what these errors means from the command interpreter.

`Wrong version of COMMAND (.BADCOM, 08Fh)`

`COMMAND2.COM` loaded its transient part from disk but its checksum was not what was expected.

`Unrecognized command (.BADCM, 08Eh)`

A given command was not an internal command and a `.COM` or `.BAT` file was not found with the same name.

`Command too long (.BUFUL, 08Dh)`

The command in a batch file exceeded 127 characters in length.

`(.OKCMD, 08Ch)`

An internal error used after executing a command passed to `COMMAND2.COM` on the command line. (There is no message for this error code.)

`Invalid parameter (.IPARM, 08Bh)`

The parameter to a command was invalid in some way eg. a number out of range.

`Too many parameters (.INP, 08Ah)`

After parsing all the parameters required for a command, there were still more non-separator characters on the command line.

`Missing parameter (.NOPAR, 089h)`

Where a parameter was expected the end of line was found.

`Invalid option (.IOPT, 088h)`

The letter given after a / on the command line was invalid for that command.

`Invalid number (.BADNO, 087h)`

Non-digit characters appeared where a number was expected.

`File for HELP not found (.NOHELP, 086h)`

The help file was not found or the parameter was not a valid `HELP` parameter.

`Wrong version of MSX-DOS (.BADVER, 085h)`

This error is never used by the command interpreter, it has its own internal message for this error. However it is provided for transient programs which may find it useful to return this error.

`Cannot concatenate destination file (.NOCAT, 084h)`

The destination file in `CONCAT` is matched by the source specification.

`Cannot create destination file (.BADEST, 083h)`

In `COPY`, creating the destination file would overwrite one of the source files (or another file that is already in use).

`File cannot be copied onto itself (.COPY, 082h)`

In `COPY`, the destination file if created would overwrite the source file.

`Cannot overwrite previous destination file (.OVDEST, 081h)`

In `COPY`, an ambiguous source was specified with a non-ambiguous, non-directory, non-device destination.

Chapter 3

Function specification

This chapter describes in detail the MSX-DOS function calls provided by MSX-DOS version 2.20.

3.1 Introduction

This document describes in detail each of the MSX-DOS 2 function calls. It should be read in conjunction with chapter 2 which describes system features such as file handles, fileinfo blocks and environment strings in general terms.

There are two ways of doing MSX-DOS function calls, reflecting the two different environments (MSX-DOS and disk BASIC) in which the system can run. Transient programs running in the MSX-DOS environment must access the functions with a "`CALL 00005h`" instruction. Disk BASIC and other MSX programs running in the disk BASIC environment (usually executing from ROM) must access the system via a "`CALL 0F37Dh`" instruction.

There are some limitations when calling the system via `0F37Dh`, particularly to do with error handling and abort routines. Also no parameters may be passed in page-1, unless they are in the master disk ROM (as they will be for disk BASIC) since the master disk ROM will be paged into page-1 when such a function call is made. The individual function descriptions mention the differences for particular functions.

3.2 List of functions

Below there is a complete list of the functions calls. "`CPM`" indicates that the function is compatible with the equivalent CP/M 2.2 function, "`MSX1`" indicates compatibility with MSX-DOS version 1, and "`NEW`" indicates a function which is new to this system. An asterisk ("`*`") indicates that the function may be safely called from a user disk error routine (see function `64h` and function `70h`).

List of MSX-DOS 2 function calls:

```
CPM MSX1   00h - Program terminate
CPM MSX1 * 01h - Console input
CPM MSX1 * 02h - Console output
CPM MSX1 * 03h - Auxiliary input
CPM MSX1 * 04h - Auxiliary output
CPM MSX1 * 05h - Printer output
CPM MSX1 * 06h - Direct console I/O
    MSX1 * 07h - Direct console input
    MSX1 * 08h - Console input without echo
CPM MSX1 * 09h - String output
CPM MSX1 * 0Ah - Buffered line input
CPM MSX1 * 0Bh - Console status
CPM MSX1 * 0Ch - Return version number
CPM MSX1   0Dh - Disk reset
CPM MSX1   0Eh - Select disk
CPM MSX1   0Fh - Open file (FCB)
CPM MSX1   10h - Close file (FCB)
CPM MSX1   11h - Search for first entry (FCB)
CPM MSX1   12h - Search for next entry (FCB)
CPM MSX1   13h - Delete file (FCB)
CPM MSX1   14h - Sequential read (FCB)
CPM MSX1   15h - Sequential write (FCB)
CPM MSX1   16h - Create file (FCB)
CPM MSX1   17h - Rename file (FCB)
CPM MSX1 * 18h - Get login vector
CPM MSX1 * 19h - Get current drive
CPM MSX1   1Ah - Set disk transfer address
    MSX1   1Bh - Get allocation information
           1Ch - Unused
           1Dh - Unused
           1Eh - Unused
           1Fh - Unused
           20h - Unused
CPM MSX1   21h - Random read (FCB)
CPM MSX1   22h - Random write(FCB)
CPM MSX1   23h - Get file size (FCB)
CPM MSX1   24h - Set random record (FCB)
           25h - Unused
    MSX1   26h - Random block write (FCB)
    MSX1   27h - Random block read (FCB)
CPM MSX1   28h - Random write with zero fill (FCB)
           29h - Unused
    MSX1 * 2Ah - Get date
    MSX1 * 2Bh - Set date
    MSX1 * 2Ch - Get time
    MSX1 * 2Dh - Set time
```

```
    MSX1 * 2Eh - Set/reset verify flag
    MSX1 * 2Fh - Absolute sector read
    MSX1 * 30h - Absolute sector write
NEW      * 31h - Get disk parameters
           32h - \
           . .   \ Unused
           . .   /
           3Fh - /
NEW        40h - Find first entry
NEW        41h - Find next entry
NEW        42h - Find new entry
NEW        43h - Open file handle
NEW        44h - Create file handle
NEW        45h - Close file handle
NEW        46h - Ensure file handle
NEW        47h - Duplicate file handle
NEW        48h - Read from file handle
NEW        49h - Write to file handle
NEW        4Ah - Move file handle pointer
NEW        4Bh - I/O control for devices
NEW        4Ch - Test file handle
NEW        4Dh - Delete file or subdirectory
NEW        4Eh - Rename file or subdirectory
NEW        4Fh - Move file or subdirectory
NEW        50h - Get/set file attributes
NEW        51h - Get/set file date and time
NEW        52h - Delete file handle
NEW        53h - Rename file handle
NEW        54h - Move file handle
NEW        55h - Get/set file handle attributes
NEW        56h - Get/set file handle date and time
NEW      * 57h - Get disk transfer address
NEW      * 58h - Get verify flag setting
NEW        59h - Get current directory
NEW        5Ah - Change current directory
NEW        5Bh - Parse pathname
NEW        5Ch - Parse filename
NEW      * 5Dh - Check character
NEW        5Eh - Get whole path string
NEW        5Fh - Flush disk buffers
NEW        60h - Fork a child process
NEW        61h - Rejoin parent process
NEW        62h - Terminate with error code
NEW        63h - Define abort exit routine
NEW        64h - Define disk error handler routine
NEW      * 65h - Get previous error code
```

```
NEW      * 66h - Explain error code
NEW        67h - Format a disk
NEW        68h - Create or destroy RAM disk
NEW        69h - Allocate sector buffers
NEW      * 6Ah - Logical drive assignment
NEW      * 6Bh - Get environment item
NEW      * 6Ch - Set environment item
NEW      * 6Dh - Find environment item
NEW      * 6Eh - Get/set disk check status
NEW      * 6Fh - Get MSX-DOS version number
NEW      * 70h - Get/set redirection status
```

3.3 Function by function definitions

Below are detailed descriptions of each of the MSX-DOS functions including both the old and new ones. The names in brackets after the function numbers are the public labels for the function codes which are defined in "CODES.MAC". Programs should use these names whenever possible.

Many of the functions below 40h return an error flag rather than an error code. If the error flag is set then the actual error code indicating the cause of the error can be obtained by the "get previous error code" function (function 65h). All of the functions above 40h return an error code in register A. Chapter 1 describes the general errors which can be returned from many of the functions. The individual function specifications here describe the main error conditions which are specific to particular functions.

Note that many of the function calls which modify the information on a disk do not automatically flush disk buffers and so the disk is not necessarily correctly updated immediately after the function call is made. Such calls include all types of "create", "write", "delete", "rename", "change file attributes" and "change file date and time" function calls. The only functions which always flush disk buffers are "flush buffers", "close" and "ensure". After these operations the disk will always be correctly updated.

3.3.1 Program terminate (00h)

- Parameters

  C = 00H (_TERM0)

- Results

  Does not return

This function terminates program with a zero return code. It is provided for compatibility with MSX-DOS 1 and CP/M, the preferred method of exiting a program is to use the "terminate with error code" function call (function 62h),

passing a zero error code if that is what is desired. See the description of that function call, and also chapter 2, for details of what happens when a program terminates. This function call never returns to the caller.

3.3.2 Console input (01h)

- Parameters

```
C = 01H (_CONIN)
```

- Results

```
L = A = Character from keyboard
```

A character will be read from the standard input (file handle 0 - usually the keyboard) and echoed to the standard output (file handle 1 - usually the screen). If no character is ready then this function will wait for one. Various control characters, as specified for the "console status" function (function 0Bh), will be trapped by this function for various control purposes. If one of these characters is detected then it will be processed and this function will wait for another character. Thus these characters will never be returned to the user by this function.

3.3.3 Console output (02h)

- Parameters

```
C = 02H (_CONOUT)
E = Character to be output
```

- Results

```
None
```

The character passed in register E is written to the standard output (file handle 1 - usually the screen). If printer echo is enabled then the character is also written to the printer. Various control codes and escape sequences are interpreted as screen control codes. A list of these is included in chapter 2, they are a sub-set of the standard VT-52 control codes. TABs will be expanded to every eighth column.

A console input status check is done, and if any of the special control characters described for the "console status" function (function 0Bh) is found then it will be processed as for that function. Any other character will be saved internally for a later "console input" function call.

3.3.4 Auxiliary input (03h)

- Parameters

```
C = 03H (_AUXIN)
```

- Results

```
L = A = Input character
```

A character is read from the auxiliary input device (file handle 3) and if no character is ready then it will wait for one. The auxiliary input device must have been installed before this function may be used. If no such device has been installed then this function will always return the end of file character ("CTRL-Z").

3.3.5 Auxiliary output (04h)

- Parameters

```
C = 04H (_AUXOUT)
E = Character to be output
```

- Results

```
None
```

The character passed in register E will be written to the auxiliary output device (file handle 3). The auxiliary output device must have been installed before this function may be used. If no such device has been installed then this function will simply throw the character away.

3.3.6 Printer output (05h)

- Parameters

```
C = 05H (_LSTOUT)
E = Character to be output
```

- Results

```
None
```

The character passed in register E will be sent to the standard printer device (file handle 4 - usually the parallel printer). The same channel is used for console output which is echoed to the printer. TABs are not expanded by this function, although they are expanded when screen output is echoed to the printer with "CTRL-P".

3.3.7 Direct console I/O (06h)

- Parameters

```
C = 06H (_DIRIO)
E = 00H...FEH - character for output
    FFH       - requests input
```

- Results

```
A = L = input - 00H - no character ready
                      else input character
        undefined for output
```

If E=FFh on entry then the keyboard will be examined for a characterfrom the standard input (file handle 0) and 00h returned if no character is ready. If a character is ready then it will be read from the standard input (file handle 0) and returned in register A without being echoed and with no check for control characters.

If E!=FFh on entry then the character in register E will be printed directly to the standard output (file handle 1) with no TAB expansion or printer echo. Also no console status check is done by this function. Note that although this function does not expand TABs, the VT-52 control codes include TAB expansion so the effect on the screen is the same.

3.3.8 Direct console input (07h)

- Parameters

```
C = 07H (_DIRIN)
```

- Results

```
L = A = Input character
```

This function is identical to the input option of function 06h, except that if no character is ready it will wait for one. Like function 06h, no echo or control characters checks will be done. This function is not compatible with CP/M which uses this function number for "get I/O byte".

3.3.9 Console input without echo (08h)

- Parameters

```
C = 08H (_INNOE)
```

- Results

```
L = A = Input character
```

This function is identical to function 01h except that the input character will not be echoed to the standard output. The same control character checks will be done. This function is not compatible with CP/M which uses this function number for "set I/O byte".

3.3.10 String output (09h)

- Parameters

```
C  = 09H (_STROUT)
DE = Address of string
```

- Results

```
None
```

The characters of the string pointed to by register DE will be output using the normal console output routine (function call 02h). The string is terminated by "$" (ASCII 24h).

3.3.11 Buffered line input (0Ah)

- Parameters

```
C  = 0AH (_BUFIN)
DE = Address of an input buffer
```

- Results

```
None
```

DE must point to a buffer to be used for input. The first byte of this buffer must contain the number of characters which the buffer can hold (0...255). A line of input will be read from the standard input device (file handle 0 - usually the keyboard) and stored in the buffer. The input will be terminated when a CR is read from the standard input. The number of characters entered, which does not include the CR itself, will be stored at (DE+1). If there is room in the buffer then the CR will be stored after the last character.

When inputting from the keyboard (which will normally be the case), a simple line editor is provided, and also a 256 byte circular buffer of previous lines which can be edited and re-entered. The details of these editing facilities are described in chapter 1, so they are not included here. When the input buffer becomes full, the console bell will be rung for each character typed which cannot be put in the buffer. Each character entered will be echoed to the standard output and also to the printer if printer echo is enabled.

3.3.12 Console status (0Bh)

- Parameters

```
C = 0BH (_CONST)
```

- Results

```
L = A = 00H if no key was pressed
      = FFH if a key was pressed
```

A flag is returned in register A to indicate whether a character is ready (that is, a key was pressed) for input from the keyboard. If a character is ready then it will be read and tested for certain special control characters. If it is not one of these then it is stored in an internal single byte buffer and subsequent call to this function will return "character ready" immediately without checking the keyboard. If this function says that a character is ready then the character may be read by function 02h or 08h.

If the character is "CTRL-C" then the program will be terminated with a ".CTRLC" error via the user's abort routine if one is defined. If the character is "CTRL-P" then printer echo will be enabled and it will be disabled if it is "CTRL-N". If the character is "CTRL-S" then the routine will hang up waiting for another character to be pressed and then return "no character ready", thus providing a "wait" facility. The character typed to continue operation will be ignored, except that of it is "CTRL-C" then the program will be terminated. These same input checks are also done for functions 01h, 02h, 08h, 09h and 0Ah.

3.3.13 Return version number (0Ch)

- Parameters

```
C = 0CH (_CPMVER)
```

- Results

```
L = A = 22H
H = B = 00H
```

This function simply returns the CP/M version number which is being emulated. This is always version 2.2 in current systems.

3.3.14 Disk reset (0Dh)

- Parameters

```
C = 0DH (_DSKRST)
```

- Results

```
None
```

Any data which is waiting in internal buffers is written out to disk. It is not necessary to call this function in order to allow a disk change as is the case with CP/M. The disk transfer address is also set back to its default value of `80h` by this function.

3.3.15 Select disk (0Eh)

- Parameters

```
C = 0EH (_SELDSK)
E = Drive number. 0=A: 1=B: etc.
```

- Results

```
L = A = Number of drives (1...8)
```

This function simply selects the specified drive as the default drive. The current drive is also stored at address `00004h` for CP/M compatibility. The number of drives available is returned in register `A` but this will not include the RAM disk.

3.3.16 Open file [FCB] (0Fh)

- Parameters

```
C  = 0FH (_FOPEN)
DE = Pointer to unopened FCB
```

- Results

```
L = A = 0FFH if file not found
      = 000h if file is found
```

The unopened `FCB` must contain a drive which may be zero to indicate the current drive and a filename and extension which may be ambiguous. The current directory of the specified drive will be searched for a matching file and if found it will be opened. Matching entries which are sub-directories or system files will be ignored, and if the filename is ambiguous then the first suitable matching entry will be opened.

Device names may be put in the `FCB` (without a colon) to allow devices to be accessed as if they were actually disk files. The standard device names are defined in chapter 2.

The low byte of the extent number is not altered by this function, and a file will only be opened if it is big enough to contain the specified extent. Normally the transient program will set the extent number to zero before calling this function. The high byte of the extent number will be set to zero to ensure compatibility with CP/M.

The filename and extension in the FCB will be replaced by the actual name of the file opened from the directory entry. This will normally be the same as what was there before but may be different if an ambiguous filename or one with lower case letters in was used.

The record count will be set to the number of 128 byte records in the specified extent, which is calculated from the file size. The file size field itself, the volume-id and the 8 reserved bytes will also be set up. The current record and random record fields will not be altered by this function, it is the application program's responsibility to initialize them before using the read or write functions.

If the file cannot be found, then the "APPEND" environment item will be examined. If this is set then it is interpreted as a drive/path string which specifies a second directory in which to look for the file. The specified directory will be searched for the file and if found it will be opened as above. In this case the drive byte of the FCB will be set to the drive on which the file was found to ensure correct accessing of the file if the original drive byte was zero (default).

3.3.17 Close file [FCB] (10h)

- Parameters

```
C = 10H (_FCLOSE)
DE = Pointer to opened FCB
```

- Results

```
L = A = 0FFH if not successful
      = 0 if successful
```

The FCB must have previously been opened with either an OPEN or a CREATE function call. If the file has only been read then this function does nothing. If the file has been written to then any buffered data will be written to disk and the directory entry updated appropriately. The file may still be accessed after a close, so the function can be regarded as doing an "ensure" function.

3.3.18 Search for first [FCB] (11h)

- Parameters

```
C = 11H (_SFIRST)
DE = Pointer to unopened FCB
```

- Results

```
L = A = 0FFH if file not found
      = 0 if file found
```

This function searches the current directory of the drive specified in the FCB (default drive if FCB contains zero) for a file which matches the filename and extension in the FCB. The filename may be ambiguous (containing "?" characters) in which case the first match will be found. The low byte of the extent field will be used, and a file will only be found if it is big enough to contain this extent number. Normally the extent field will be set to zero by the program before calling this function. System file and sub-directory entries will not be found.

If a suitable match is found (A=0) then the directory entry will be copied to the DTA address, preceded by the drive number. This can be used directly as an FCB for an OPEN function call if desired. The extent number will be set to the low byte of the extent from the search FCB, and the record count will be initialized appropriately (as for OPEN). The attributes byte from the directory entry will be stored in the S1 byte position, since its normal position (immediately after the filename extension field) is used for the extent byte.

If no match is found (A=0FFh) then the DTA will not be altered. In no case will the FCB pointed to by DE be modified at all. This function remembers sufficient information internally to allow it to continue the search with a SEARCH FOR NEXT function, so it is not necessary for the FCB to be preserved if doing a SEARCH FOR NEXT function.

In CP/M, if the drive number is set to "?" in this function then all directory entries, allocated or free will be matched. Also if the extent field is set to "?" then any extent of a file will be matched. Both of these features are normally only used by special purpose CP/M programs which are generally specific to the CP/M filing system (such as "STAT"). Neither feature is present in MSX-DOS 1/2.

3.3.19 Search for next [FCB] (12h)

- Parameters

```
C = 12H (_SNEXT)
```

- Results

```
L = A = 0FFH if file not found
      = 0 if file found
```

It continues the search to look for the next match with the filename. The results returned from this function are identical to SEARCH FOR FIRST and all the same comments apply. The information used to continue the search is held internally within MSX-DOS and so the original FCB used in the SEARCH FOR FIRST need not still exist.

3.3.20 Delete file [FCB] (13h)

- Parameters

```
C = 13H (_FDEL)
DE = Pointer to unopened FCB
```

- Results

```
L = A = 0FFH if no files deleted
```

All files in the current directory of the disk specified by the FCB, and which match the ambiguous filename in the FCB, are deleted. Sub-directories, system files, hidden files and read only files are not deleted. If any files at all are successfully deleted then this function returns with A=0. A return with A=FFh indicates that no files were deleted.

3.3.21 Sequential read [FCB] (14h)

- Parameters

```
C = 14H (_RDSEQ)
DE = Pointer to opened FCB
```

- Results

```
L = A = 01H if error (end of file)
      = 0 if read was successful
```

This function reads the next sequential 128 byte record from the file into the current disk transfer address. The record is defined by the current extent (high and low bytes) and the current record. After successfully reading the record, this function increments the current record and if it reaches 080h, sets it back to zero and increments the extent number. The record count field is also kept updated when necessary.

Unlike CP/M it is possible to have partially filled records, since the file size is not necessarily a multiple of 128 bytes. If this occurs then the partial record is padded out with zeroes when it is copied to the transient program's DTA address.

3.3.22 Sequential write [FCB] (15h)

- Parameters

```
C = 15H (_WRSEQ)
DE = Pointer to opened FCB
```

- Results

```
L = A = 01H if error (disk full)
      = 0 if write was successful
```

This function writes the 128 bytes from the current disk transfer address to the file at the position defined by the current record and extent, which are then incremented appropriately. The record count byte is kept updated correctly if the file is extended or if the write moves into a new extent. The file size in the FCB is also updated if the file is extended.

3.3.23 Create file [FCB] (16h)

- Parameters

```
C = 16H (_FMAKE)
DE = Pointer to unopened FCB
```

- Results

```
L = A = 0FFH if unsuccessful
      = 0 if successful
```

This function creates a new file in the current directory of the specified drive and opens it ready for reading and writing. The drive, filename and low byte of the extent number must be set up in the FCB and the filename must not be ambiguous. Checks will be done to ensure that invalid filenames are not created.

If there is already a file of the required name then the action depends on the value of the extent number byte. Normally this will be zero and in this case the old file will be deleted and a new one created. However if the extent number is non-zero then the existing file will be opened without creating a new file. This ensures compatibility with early versions of CP/M where each extent had to be explicitly created.

In all cases the resulting file will be opened with the required extent number exactly as if an OPEN function call had been done.

3.3.24 Rename file [FCB] (17h)

- Parameters

```
C = 17H (_FREN)
DE = Pointer to unopened FCB
```

- Results

```
L = A = 0FFH not if successful
      = 0 if successful
```

The unopened FCB has the normal drive and filename, and also a second filename starting at (DE+17). Every file in the current directory of the specified drive which matches the first filename, is changed to the second filename with "?" characters in the second filename leaving the appropriate character unchanged. Checks are done to prevent duplicate or illegal filenames from being created. Entries for sub-directories, hidden files and system files will not be renamed.

3.3.25 Get login vector (18h)

- Parameters

```
C = 18H (_LOGIN)
```

- Results

```
HL = Login vector
```

This function returns a bit set in HL for each drive which is available, bit-0 of L corresponding to drive "A:". Up to eight drives ("A:" to "H:") are supported by the system currently, so register H will usually be zero on return.

3.3.26 Get current drive (19h)

- Parameters

```
C = 19H (_CURDRV)
```

- Results

```
L = A = Current drive (0=A: etc)
```

This function just returns the current drive number.

3.3.27 Set disk transfer address (1Ah)

- Parameters

```
C = 1AH (_SETDTA)
DE = Required Disk Transfer Address
```

- Results

```
None
```

This function simply records the address passed in DE as the disk transfer address. This address will be used for all subsequent FCB read and write calls, for "search for first" and "search for next" calls to store the directory entry, and for absolute read and write calls. It is not used by the new MSX-DOS read and write functions. The address is set back to 80h by a DISK RESET call.

3.3.28 Get allocation information (1Bh)

- Parameters

```
C = 1BH (_ALLOC)
E = Drive number (0=current, 1=A: etc)
```

- Results

```
A = Sectors per cluster
BC = Sector size (always 512)
DE = Total clusters on disk
HL = Free clusters on disk
IX = Pointer to DPB
IY = Pointer to first FAT sector
```

This function returns various information about the disk in the specified drive. It is not compatible with CP/M which uses this function number to return the address of an allocation vector. Note that unlike MSX-DOS 1, only the first sector of the FAT may be accessed from the address in IY, and the data there will only remain valid until the next MSX-DOS call.

3.3.29 Random read [FCB] (21h)

- Parameters

```
C = 21H (_RDRND)
DE = Pointer to opened FCB
```

- Results

```
L = A = 01H if error (end of file)
      = 0 if read was successful
```

This function reads a 128 byte record from the file to the current disk transfer address. The file position is defined by the three byte random record number in the FCB (bytes 21h...23h). Unlike CP/M all three bytes of the random record number are used. A partial record at the end of the file will be padded with zeroes before being copied to the user's DTA.

The random record number is not altered so successive calls to this function will read the same record unless the transient program alters the random record number. A side effect is that the current record and extent are set up to refer to the same record as the random record number. This means that sequential reads (or writes) can follow a random read and will start from the same record. The record count byte is also set up correctly for the extent.

3.3.30 Random write [FCB] (22h)

- Parameters

```
C = 22H (_WRRND)
DE = Pointer to opened FCB
```

- Results

```
L = A = 01H if error (disk full)
      = 0 if no error
```

This function writes a 128 byte record from the current disk transfer address to the file, at the record position specified by the three byte random record number (bytes 21h...23h). All three bytes of the random record number are used. If the record position is beyond the current end of file then un-initialized disk space will be allocated to fill the gap.

The random record number field will not be changed, but the current record and extent fields will be set up to refer to the same record. The record count byte will be adjusted as necessary if the file is being extended or if the write goes into a new extent.

3.3.31 Get file size [FCB] (23h)

- Parameters

```
C = 23H (_FSIZE)
DE = Pointer to unopened FCB
```

- Results

```
L = A = 0FFH if file not found
      = 0 if file found OK
```

This function searches for the first match with the filename in the FCB, exactly the same as OPEN FILE (function 0FH). The size of the located file is rounded up to the nearest 128 bytes and the number of records determined. The three byte random record field of the FCB is set to the number of records, so it is the number of the first record which does not exist. The fourth byte of the random record number is not altered.

3.3.32 Set random record [FCB] (24h)

- Parameters

```
C = 24H (_SETRND)
DE = Pointer to opened FCB
```

- Results

```
None
```

This function simply sets the three byte random record field in the FCB to the record determined by the current record and extent number. The fourth byte of the random record number is not altered. No check is done as to whether the record actually exists in the file.

3.3.33 Random block write [FCB] (26h)

- Parameters

```
C = 26H (_WRBLK)
DE = Pointer to opened FCB
HL = Number of records to write
```

- Results

```
A = 01H if error
  = 0 if no error
```

Data is written from the current disk transfer address to the position in the file defined by the random record number. The record size is determined by the record size field in the FCB (bytes 0Eh and 0Fh) which must be set by the user after opening the file and before calling this function. If the record size is less than 64 bytes then all four bytes of the random record number are used, otherwise only the first three are used.

The number of records to be written is specified by HL, and together with the record size this determines the amount of data to be written. An error will be returned if the size exceeds 64k, thus limiting the maximum size of a transfer.

After writing the data, the random record field is adjusted to the next record number in the file (ie. HL is added on to it). The current record and extent fields are not used or altered. The file size field is updated if the file has been extended.

The record size can be any value from 1...0FFFFh. Small record sizes are no less efficient that large record sizes so if desired the record size can be set to one and the record count then becomes a byte count. It is desirable to write as much as possible with one function call since one large transfer will be quicker than several small ones.

If the number of records to write (HL) is zero then no data will be written, but the size of the file will be altered to the value specified by the random record field. This may be either longer or shorter than the file's current size and disk space will be allocated or freed as required. Additional disk space allocated in this way will not be initialized to any particular value.

3.3.34 Random block read [FCB] (27h)

- Parameters

```
C = 27H (_RDBLK)
DE = Pointer to opened FCB
HL = Number of records to read
```

- Results

```
A = 01H if error (usually caused by end-of-file)
  = 0 if no error
HL = Number of records actually read
```

This function is the complement of the BLOCK WRITE function described above and most of the same comments apply as regards its use. Again if large blocks are read then it will be much faster than the normal CP/M operation.

For example if it is desired to read 20k from a file, it is better to read the 20k with one function call rather than 20 separate function calls of 1k each. However it makes no difference whether the 20k read is done with a record size of 1 and a record count of 20k, with a record size of 20k and a record count of 1, or any intermediate combination.

The number of records actually read is returned in HL. This may be smaller than the number of records requested if the end of the file was encountered. In this case any partial record will be padded out with zeroes before being copied to the users DTA. The random record field is adjusted to the first record not read, ie. the value returned in HL is added on to it.

3.3.35 Random write with zero fill [FCB] (28h)

- Parameters

```
C = 28H (_WRZER)
DE = Pointer to opened FCB
```

- Results

```
L = A = 01H if error
      = 00H if no error
```

This function is identical to RANDOM WRITE (function 22h) except that if the file has to be extended, any extra allocated disk clusters will be filled with zeroes before writing the data.

3.3.36 Get date (2Ah)

- Parameters

```
C = 2AH (_GDATE)
```

- Results

```
HL = Year 1980...2079
D = Month (1=Jan...12=Dec)
E = Date (1...31)
A = Day of week (0=Sun...6=Sat)
```

This function simply returns the current value of the internal calender in the format shown.

3.3.37 Set date (2Bh)

- Parameters

```
C = 2BH (_SDATE)
HL = Year 1980...2079
D = Month (1=Jan...12=Dec)
E = Date (1...31)
```

- Results

```
A = 00H if date was valid
  = FFH if date was invalid
```

The supplied date is checked for validity and if it is valid then it is stored as the new date. The validity checks include full checking for the number of days in each month and leap years. If the date is invalid then the current date will be left unaltered. The date is stored in the real time clock chip so it will be remembered when the machine is turned off.

3.3.38 Get time (2Ch)

- Parameters

```
C = 2CH (_GTIME)
```

- Results

```
H = Hours (0...23)
L = Minutes (0...59)
D = Seconds (0...59)
E = Centiseconds (always zero)
```

This function returns the current value of the system clock in the format shown.

3.3.39 Set time (2Dh)

- Parameters

```
C = 2DH (_STIME)
H = Hours (0...23)
L = Minutes (0...59)
D = Seconds (0...59)
E = Centiseconds (ignored)
```

- Results

```
A = 00H if time was valid
  = FFH if time was invalid
```

This function sets the internal system clock to the specified time value. If the time is invalid then register `A` will be returned as `0FFh` to indicate an error and the current time will be left unaltered. The time is stored in the real time clock chip and so it will be remembered and kept correct when the machine is turned off.

3.3.40 Set/reset verity flag (2Eh)

- Parameters

```
C = 2EH (_VERIFY)
E = 0 to disable verify
  != 0 to enable verify
```

- Results

```
None
```

This function simply enables or disables automatic verification of all writes. It defaults to off when MSX-DOS is started up. Enabling verify improves system reliability but also slows down write operations. Note that this function depends on the disk driver and the verification will not be done if the driver does not support it.

3.3.41 Absolute sector read (2Fh)

- Parameters

```
C = 2FH (_RDABS)
DE = Sector number
L = Drive number (0=A: etc.)
H = Number of sectors to read
```

- Results

```
A = Error code (0=> no error)
```

This function reads sectors directly from the disk without interpreting them as files. The disk must be a valid DOS disk in order for the sector number to be translated into a physical position on the disk. The sectors will be read to the current disk transfer address. Any disk error will be reported by the system in the usual way.

3.3.42 Absolute sector write (30h)

- Parameters

```
C = 30H (_WRABS)
DE = Sector number
L = Drive number (0=A: etc.)
H = Number of sectors to write
```

- Results

```
A = Error code
```

This function writes sectors directly to the disk without interpreting them as files. The disk must be a valid DOS disk in order for the sector number to be translated into a physical position on the disk. The sectors will be written from the current disk transfer address. Any disk errors are reported by the system in the usual way.

3.3.43 Get disk parameters (31h)

- Parameters

```
C = 31H (_DPARM)
DE = Pointer to 32 byte buffer
L = Drive number (0=default, 1=A: etc.)
```

- Results

```
A = Error code
DE = Preserved
```

This functions returns a series of parameters relating to the format of the disk in the specified drive, to the buffer allocated within the user's program. It is useful for programs which are going to do absolute sector reads and writes, in order for them to be able to interpret the absolute sector numbers. The parameters returned contain some redundant information in order to provide parameters which are most useful to transient programs. The format of the returned parameter block is:

DE+0	Physical drive number (1=A: etc)
DE+1,2	Sector size (always 512 currently)
DE+3	Sectors per cluster (non-zero power of 2)
DE+4,5	Number of reserved sectors (usually 1)
DE+6	Number of copies of the FAT (usually 2)
DE+7,8	Number of root directory entries
DE+9,10	Total number of logical sectors
DE+11	Media descriptor byte
DE+12	Number of sectors per FAT
DE+13..14	First root directory sector number
DE+15..16	First data sector number
DE+17..18	Maximum cluster number

DE+19 Dirty disk flag

DE+20..23 Volume id. (-1 => no volume id.)

DE+24..31 Reserved (currently always zero)

The dirty disk flag indicates whether in the disk there is a file which can be recovered by UNDEL command. It is reset when the file allocation is done.

3.3.44 Find first entry (40h)

- Parameters

```
C = 40H (_FFIRST)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
HL = filename ASCIIZ string (only if DE = fileinfo pointer)
B = Search attributes
IX = Pointer to new fileinfo block
```

- Results

```
A = Error
(IX) = Filled in with matching entry
```

The "drive/path" portion of the string, or the fileinfo block, specifies a directory which is to be searched. A ".IATTR" error will be returned if a fileinfo block which specifies a file is passed. The "file" portion of the string, or the filename ASCIIZ string in HL, determines what filenames will be matched. If no match is found then a ".NOFIL" error is returned, otherwise the fileinfo block pointed to by IX is filled in with the details of the matching entry.

The filename may contain ambiguous filename characters ("?" or "*") in which case the first matching entry will be returned. If the filename is null (either the ASCIIZ string pointed to by DE is null or ends in a "\" or the filename string pointed to by HL is null), then this function will behave exactly as if the filename was "*.*" so any name will match.

The attributes byte in register B specifies what type of entry will be matched. If it is zero then only non-hidden, non-system files will be found. If the directory, hidden or system bits in register B are set then entries with these attributes will be matched as well as ordinary files. The read only and archive bits of register B are ignored.

If the volume name bit of register B is set then the search is exclusive, only the volume label entry will be found. In this case also the fileinfo block and filename or the drive/path/file string are ignored apart from specifying the drive. This means that the volume name will always be found in the root directory if it exists whether or not it matches the filename given.

If DE points to a fileinfo block, then if desired, IX can point to the same fileinfo block. In this case when a match is found the new fileinfo block will overwrite the old one.

3.3.45 Find next entry (41h)

- Parameters

```
C = 41H (_FNEXT)
IX = Pointer to fileinfo block from previous find first function
```

- Results

```
A = Error
(IX) = Filled in with next matching entry
```

This function should only be used after a "find first entry" function call. It searches the directory for the next match to the (presumably ambiguous) filename which was given to the "find first entry" function call. If there are no more matching entries then a ".NOFIL" error is returned, otherwise the fileinfo block is filled in with the information about the new matching entry.

3.3.46 Find new entry (42h)

- Parameters

```
C = 42H (_FNEW)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
HL = filename ASCIIZ string (only if DE = fileinfo pointer)
B = b0..b6 = Required attributes
    b7 = Create new flag
IX = Pointer to new fileinfo block containing template filename
```

- Results

```
A = Error
(IX) = Filled in with new entry
```

This function is very similar to the "find first entry" function described above. The parameters in HL and DE are used in exactly the same way to specify a directory entry. However instead of searching the selected directory for an entry which matches the specified name, a new entry will be created with this name. The fileinfo block pointed to by IX will be filled in with information about the new entry just as if it had been found with a "find first entry" call.

If there are any ambiguous characters ("?" or "*") in the filename, then they will be replaced by the appropriate character from a "template filename" in the filename position of the new fileinfo block pointed to by IX. If the result is still ambiguous, or otherwise illegal, then a ".IFNM" error is returned. This is useful for copy operations which do an automatic rename.

Like "find first entry", if the filename is null, then it will be treated exactly as if it was "*.*". For this function that means that the template filename will be used as the new filename to create.

A ".DRFUL" error will be returned if there is no room in the root directory, and a ".DKFUL" if a sub-directory must be extended and the disk is already full.

The attribute byte passed in register B is the attribute which the new entry will be given. If the volume name bit is set then a volume name will be created in the root directory. If the directory bit is set then the entry created will be for a sub-directory, otherwise it will be for a file. The system, hidden and read only bits may be set for a file, and the hidden bit for a sub-directory. A file will always be created with the archive attribute bit set.

A file will be created as zero length with the current date and time. A sub-directory will have a single cluster allocated to it and the "." and ".." entries will be initialized appropriately.

If there is already an entry with the specified name in the directory then the action depends on the "create new" flag (bit-7 of register B) and also on the type of the entry. If the "create new" flag is set then a ".FILEX" error will always be returned. Setting this flag thereforeensures that an existing file will not be deleted.

If an entry already exists and the "create new" flag is not set then the type of the existing entry is examined to see whether it can be deleted to make room for the new file. An error will be returned if the entry is a read only file (".FILRO" error), a system file (".SYSX" error) or a sub-directory (".DIRX" error) or there is a file handle already open to this file (".FOPEN" error). If we are trying to create a sub-directory then even an ordinary file will not be deleted (".FILEX" error).

For all of these error codes (".FILEX", ".FILRO", ".SYSX", ".DIRX", ".FOPEN"), the fileinfo block will be filed in with the details of the already existing entry and this fileinfo block may be used exactly as if it had been returned from a "find first" function.

3.3.47 Open file handle (43h)

- Parameters

```
C = 43H (_OPEN)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
A = Open mode. b0 set => no write
               b1 set => no read
               b2 set => inheritable
               b3..b7 - must be clear
```

- Results

```
A = Error
B = New file handle
```

The drive/path/file string or the fileinfo block should normally refer to a file rather than a sub-directory or volume name. If it is a volume name then a ".IATTR" error will be returned. If it is a sub-directory then ".DIRX" error will be returned.

Assuming that a file is specified then it will be opened ready for reading and/or writing (depending on the open mode in A) and a new file handle for it will be returned in register B. The lowest available file handle number will be used and an error will result if there are no spare file handles (".NHAND" error), or insufficient memory (".NORAM" error).

If the "no read" bit of register A is set then reads from the file handle will be rejected and if the "no write" bit is set then writes will be rejected, in both cases with an ".ACCV" error. Writes will also be rejected if the file is read only (".FILRO" error). If the "inheritable" bit of register A is set then the file handle will be inherited by a new process created by the "fork" function call (see function 60h).

If a device file handle is opened by giving a filename which matches one of the built in devices (for example "CON" or "NUL"), then it will always be opened initially in ASCII mode. The IOCTL function (function 4Bh) can be used to change this to binary mode but great care must be taken in reading from devices in binary mode because there is no end of file condition.

3.3.48 Create file handle (44h)

- Parameters

```
C = 44H (_CREATE)
DE = Drive/path/file ASCIIZ string
A = Open mode. b0 set => no write
               b1 set => no read
               b2 set => inheritable
               b3..b7 - must be clear
B = b0..b6 = Required attributes
    b7 = Create new flag
```

- Results

```
A = Error
B = New file handle
```

A file or sub-directory, as specified by the attributes in register B, will be created with the name and in the directory specified by the drive/path/file string. A ".IATTR" error is returned if register B specifies a volume name.

An error will be returned if the file or sub-directory cannot be created. The error conditions in this case are the same as for the "find new entry" function (function 42h) with the main error codes being ".FILEX", ".DIRX", ".SYSX", ".FILRO", ".FOPEN", ".DRFUL" and ".DKFUL". Like the "find new" function,

if the "create new" flag (bit-7 of register B) is set then an existing file will not be deleted and will always return a ".FILEX" error.

If the attributes byte specifies a sub-directory then the hidden bit may also be set to create a hidden sub-directory. For a file, the hidden, system or read only bits may be set to create a file with the appropriate attributes. An invalid attributes bits will simply be ignored. A file will always be created with the archive attribute bit set.

A file will automatically be opened just as for the "open" function described above, and a file handle returned in register B. The "open mode" parameter is interpreted in the same way as for the "open" function. A sub-directory will not be opened (because this is meaningless) so register B will be returned as 0FFh which can never be a valid file handle.

3.3.49 Close file handle (45h)

- Parameters

```
C = 45H (_CLOSE)
B = File handle
```

- Results

```
A = Error
```

This function releases the specified file handle for re-use. If the associated file has been written to then its directory entry will be updated with a new date and time, the archive attributes bit will be set, and any buffered data will be flushed to disk. Any subsequent attempt to use this file handle will return an error. If there are any other copies of this file handle, created by "duplicate file handle" or "fork", then these other copies may still be used.

3.3.50 Ensure file handle (46h)

- Parameters

```
C = 46H (_ENSURE)
B = File handle
```

- Results

```
A = Error
```

If the file associated with the file handle has been written to then its directory entry will be updated with a new date and time, the archive attributes bit will be set, and any buffered data will be flushed to disk. The file handle is not released and so it can still be used for accessing the file, and the current file pointer setting will not be altered.

3.3.51 Duplicate file handle (47h)

- Parameters

```
C = 47H (_DUP)
B = File handle
```

- Results

```
A = Error
B = New file handle
```

This function creates a copy of the specified file handle. The lowest available file handle number will always be used and a ".NHAND" error returned if there are none available. The new file handle will refer to the same file as the original and either one may be used. If the file pointer of one handle is moved, the other one will also be moved. If either handle is closed the other one may still be used.

Note that because duplicate file handles created by this function are not *separately opened*, they do not count as separate file handles for the purposes of generating ".FOPEN" errors. So for example a "DUP"ed file handle may be renamed (function 53h) or have its attributes changed (function 55h) and the effect will apply to both file handles. Note in particular that if one copy of a "DUP"ed file handle is deleted (function 54h) then the file really will be deleted and the other file handle, although still open, can no longer be used safely. If it is used (other than being closed, ensured or deleted) then an ".FDEL" error will be returned.

3.3.52 Read from file handle (48h)

- Parameters

```
C = 48H (_READ)
B = File handle
DE = Buffer address
HL = Number of bytes to read
```

- Results

```
A = Error
HL = Number of bytes actually read
```

The specified number of bytes are read from the file at the current file pointer position and copied to the buffer address specified in register DE. The file pointer is then updated to the next sequential byte. A ".ACCV" error will be returned if the file handle was opened with the "no read" access bit set.

The number of bytes read may be less than the number requested for various reasons, and the number read will be returned in register HL if there is no error. In general if less is read than requested then this should not be treated as an

error condition but another read should be done to read the next portion, until a ".EOF" error is returned. An ".EOF" error will never be returned for a partial read, only for a read which reads zero bytes. Reading files in this way ensures that device file handles will work correctly (see below).

For disk files the number of bytes read will only be less than the number requested if the end of the file is reached and in this case the next read operation will read zero bytes and will return an ".EOF" error. When reading from a device file handle (for example the standard file handles 0 to 4), the behaviour depends on the particular device, and on whether it is being read in ASCII or binary mode (see function 4Bh below). The "CON" device will be described as an example because it is the most commonly used device, but other devices behave similarly.

When reading from the "CON" device in binary mode, characters will be read from the keyboard, without any interpretation and without being echoed to the screen or printer. The exact number of characters requested will always be read and there is no end of file condition. Because of the lack of any end of file indication, great care must be taken when reading from devices in binary mode.

A read function call to the "CON" device in ASCII mode (the default mode and that which normally applies to the standard input channel), will only read one line of input. The input line will be read from the keyboard with the normal line editing facilities available to the user, and the character typed will be echoed to the screen and to the printer if CTRL-P is enabled. Special control characters "CTRL-P", "CTRL-N", "CTRL-S" and "CTRL-C" will be tested for and will be treated exactly as for the console status function 0Bh.

When the user types a carriage return the line will be copied to the read buffer, terminated with a CR-LF sequence and the read function will return with an appropriate byte count. The next read will start another buffered line input operation. If the number of bytes requested in the read was less than the length of the line input then as many character as requested will be returned, and the next read function call will return immediately with the next portion of the line until it has all been read.

If the user types a line which starts with a "CTRL-Z" character then this will be interpreted as indicating end of file. The line will be discarded and the read function call will read zero bytes and return an ".EOF" error. A subsequent read after this will be back to normal and will start another line input. The end of file condition is thus not permanent.

3.3.53 Write to file handle (49h)

- Parameters

```
C = 49H (_WRITE)
B = File handle
DE = Buffer address
HL = Number of bytes to write
```

- Results

```
A = Error
HL = Number of bytes actually written
```

This function is very similar to the "read" function above (function 48h). The number of bytes specified will be written to the current file pointer position in the file, and the file pointer will be adjusted to point to just after the last byte written. If the file was opened with the "no write" access bit set then a ".ACCV" error will be returned, and if the file is read only then a ".FILRO" error will be returned.

If the write goes beyond the current end of file then the file will be extended as necessary. If the file pointer is already beyond the end of the file then disk space will be allocated to fill the gap and will not be initialized. If there is insufficient disk space then a ".DKFUL" error will be returned and no data will be written, even if there was room for some of the data.

The number of bytes written can usually be ignored since it will either be zero if an error is returned or it will be equal to the number requested if the write was successful. It is very much more efficient to write files in a few large blocks rather than many small ones, so programs should always try to write in as large blocks as possible.

This function sets a "modified" bit for the file handle which ensures that when the file handle is closed or ensured, either explicitly or implicitly, the directory entry will be updated with the new date, time and allocation information. Also the archive bit will be set to indicate that this file has been modified since it was last archived.

Writing to device file handles is not a complicated as reading from them because there are no end of file conditions or line input to worry about. There are some differences between ASCII and binary mode when writing to the "CON" device, in that a console status check is done in ASCII mode only. Also printer echo if enabled will only be done in ASCII mode.

3.3.54 Move file handle pointer (4Ah)

- Parameters

```
C = 4AH (_SEEK)
B = File handle
A = Method code
DE:HL = Signed offset
```

- Results

```
A = Error
DE:HL = New file pointer
```

The file pointer associated with the specified file handle will be altered according to the method code and offset, and the new pointer value returned in DE:HL. The method code specifies where the signed offset is relative to as follows:

```
A=0 Relative to the beginning of the file
A=1 Relative to the current position
A=2 Relative to the end of the file
```

Note that an offset of zero with an method code of 1 will simply return the current pointer value, and with a method code of 2 will return the size of the file. No end of file check is done so it is quite possible (and sometimes useful) to set the file pointer beyond the end of the file. If there are any copies of this file handle created by the "duplicate file handle" function (function 47h) or the "fork" function (function 60h) then their file pointer will also be changed.

The file pointer only has any real meaning on disk files since random access is possible. On device files the file pointer is updated appropriately when any read or write is done, and can be examined or altered by this function. However changing will have no effect and examining it is very unlikely to be useful.

3.3.55 I/O control for devices (4Bh)

- Parameters

```
C = 4BH (_IOCTL)
B = File handle
A = Sub-function code
    00H -> get file handle status
    01H -> set ASCII/binary mode
    02H -> test input ready
    03H -> test output ready
    04H -> find screen size
DE = Other parameters
```

- Results

```
A = Error
DE = Other results
```

This function allows various aspects of file handles to be examined and altered. In particular it can be used to determine whether a file handle refers to a disk file or a device. This is useful for programs which want to behave differently for disk files and device I/O.

This function is passed the file handle in register B and a sub-function code in register A which specifies one of various different operations. Any other parameters required by the particular sub-function are passed in register DE and results are returned in register DE. If the sub-function code is invalid then a ".ISBFN" error will be returned.

If A=0 then the operation is "get file handle status". This returns a word of flags which give various information about the file handle. The format of this word is different for device file handles and disk file handles, and bit-7 specifies which it is. The format of the word is as follows:

For devices:

```
DE:
b0 set -> console input device
b1 set -> console output device
b2..b4 reserved
b5 set   -> ASCII mode
   clear -> binary mode
b6 set -> end of file
b7 always set (-> device)
b8..b15 reserved
```

For disk files:

```
DE:
b0..b5 drive number (0=A: etc)
b6 set -> end of file
b7 always clear (-> disk file)
b8..b15 reserved
```

Note that the end of file flag is the same for devices as for disk files. For devices it will be set if the previous attempt to read from the device produced a `".EOF"` error and will be cleared by the next read. For disk files it is worked out by comparing the file pointer with the file size.

If `A=1` then the operation is a `"set ASCII/binary mode"`. This operation is only allowed for device file handles. An ASCII/binary flag must be passed in bit-5 of register `E` (exactly where it is returned by `"get file handle status"`). This is set for ASCII mode and clear for binary mode. All other bits of register `DE` are ignored.

If `A`=2 or 3 then the operation is `"test input ready"` or `"test output ready"` respectively. In both cases a flag is returned in register `E` which is `FFh` if the file handle is ready for a character and `00h` if not. The exact meaning of `"ready for a character"` depends on the device. Disk file handles are always ready for output, and are always ready for input unless the file pointer is at the end of file. The `"CON"` device checks the keyboard status to determine whether it is ready for input or not.

If `A`=4 the the operation is `"get screen size"`. This returns the logical screen size for the file handle with the number of rows in register `D` and the number of columns in register `E`. For devices with no screen size (such as disk files) both `D` and `E` will be zero. Zero for either result should therefore be interpreted as "unlimited". For example this function is used by the `"DIR /W"` command to decide how many files to print per line, and a value of zero for register `E` is defaulted to 80.

3.3.56 Test file handle (4Ch)

- Parameters

```
C = 4CH (_HTEST)
```

```
B = File handle
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
```

- Results

```
A = Error
B = 00H -> not the same file
    FFH -> same file
```

This rather specialist function is passed a file handle and either a drive/path/file string or a fileinfo block which identifies a file. It determines if the two files are actually the same file and returns a flag indicating the result. Note that if the file handle is for a device rather than a disk file then it will always return "B=00h" to indicate "not the same file".

This function allows the "COPY" command to detect certain error conditions such as copying file onto themselves and give the user informative error messages. It may also be useful for other programs which need to do similar tests.

3.3.57 Delete file or subdirectory (4Dh)

- Parameters

```
C = 4DH (_DELETE)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
```

- Results

```
A = Error
```

This function deletes the object (file or sub-directory) specified by the drive/path/file string or the fileinfo block. Global filename characters are not allowed so only one file or sub-directory can be deleted with this function. A sub-directory can only be deleted if it is empty or an error (".DIRNE") occurs if not). The "." and ".." entries in a sub-directory cannot be deleted (".DOT" error) and neither can the root directory. A file cannot be deleted if there is a file handle open to it (.FOPEN error) or if it is read only (.FILRO error).

If it is a file then any disk space which was allocated to it will be freed. If the disk is an MSX-DOS 2 disk then enough information is retained on the disk to allow the "UNDEL" utility program do undelete the file. This information is only retain ed until the next disk space allocation (usually a write to a file) is done on this disk. After making this function call, if a fileinfo block was passed then it must not be used again (other than passing it to a "find next entry" function) since the file to which it refers no longer exists.

If a device name such as "CON" is specified then no error will be returned but the device will not actually be deleted.

3.3.58 Rename file or subdirectory (4Eh)

- Parameters

```
C = 4EH (_RENAME)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
HL = New filename ASCIIZ string
```

- Results

```
A = Error
```

This function renames the object (file or sub-directory) specified by the drive/path/file string or the fileinfo block, with the new name in the string pointed to by HL. The new filename string must not contain a drive letter or directory path (".IFNM" error if it does). If a device name such as "CON" is specified then no error will be returned but the device will not actually be renamed.

Global filename characters are not allowed in the drive/path/file string, so only one object can be renamed by this function. However global filename characters are allowed in the new filename passed in HL and where they occur the existing filename character will be left unaltered. Checks are done to avoid creating an illegal filename, for example a file called "XYZ" cannot be renamed with a new filename string of "????A" because the new filename would be "XYZ A" which is illegal. In this case a ".IFNM" error will be returned.

If there is already an entry with the new filename then an error (".DUPF") is returned to avoid creating duplicate filenames. The "." and ".." entries in a sub-directory cannot be renamed (".IDOT" error) and neither can the root directory (it has noname). A file cannot be renamed if there is a file handle open to it (".FOPEN" error) although a read only file can be renamed.

Note that if DE pointed to a fileinfo block, this is not updated with the new name of the file. Therefore care must be taken in using the fileinfo block after making this function call.

3.3.59 Move file or subdirectory (4Fh)

- Parameters

```
C = 4FH (_MOVE)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
HL = New path ASCIIZ string
```

- Results

```
A = Error
```

This function moves the object (file or sub-directory) specified by the drive/path/file string or the fileinfo block, to the directory specified by the new path string pointed to by HL. There must not be a drive name in the new path string. If a device name such as "CON" is specified then no error will be returned but the device will not actually be moved.

Global filename characters are not allowed in any of the strings so only one object (file or sub-directory) can be moved by this function, although if a sub-directory is moved, all its descendants will be moved with it. If there is already an entry of the required name in the target directory then a ".DUPF" error is returned to prevent creating duplicate filenames. The "." and ".." entries in a sub-directory cannot be moved (".DOT" error) and also a directory cannot be moved into one of its own descendants (".DIRE" error) since this would create an isolated loop in the filing system. A file cannot be moved if there is a file handle open to it (".FOPEN" error).

Note that if a fileinfo block is passed to this function, the internal information in the fileinfo block is not updated to reflect the new location of the file. This is necessary because otherwise the fileinfo block could not be used for a subsequent "find next" function call. However it does mean that the fileinfo block no longer refers to the moved file and so must not be used for any operations on it such as "rename" or "open".

3.3.60 Get/set file attributes (50h)

- Parameters

```
C = 50H (_ATTR)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
A = 0 => get attributes
    1 => set attributes
L = New attributes byte (only if A=1)
```

- Results

```
A = Error
L = Current attributes byte
```

This function is normally used to change the attributes of a file or sub-directory. It can also be used to find out the current attributes but this is more usually done with the "find first entry" function (function 40h). If A=0 then the current attributes byte for the file or sub-directory will just be returned in register L.

If A=1 then the attributes byte will be set to the new value specified in register L, and this new value will also be returned in register L. Only the system, hidden, read only and archive bits may be altered for a file, and only the hidden bit for a sub-directory. An ".IATTR" error will be returned if an attempt is made to alter any other attribute bits. If a fileinfo block is passed then the attributes byte in it will not be updated with the new setting.

Global filename characters are not allowed so only one object (file or sub-directory) can have its attributes set by this function. The attributes of the root directory cannot be changed because it does not have any. The attributes of a file cannot be changed if there is a file handle open to it (".FOPEN" error). The attributes of the "." and ".." directory entries however can be changed. If a device name such as "CON" is specified then no error will be returned but the device's attributes will not actually be changed (since it does not have any).

3.3.61 Get/set file date and time (51h)

- Parameters

```
C = 51H (_FTIME)
DE = Drive/path/file ASCIIZ string
     or fileinfo block pointer
A = 0 => get date and time
    1 => set date and time
IX = New time value (only if A=1)
HL = New date value (only if A=1)
```

- Results

```
A = Error
DE = Current file time value
HL = Current file date value
```

If A=1 then this function sets the date and time of last modification of the file or sub-directory specified by the drive/path/file string or fileinfo block. Global filename characters are not allowed in any part of the string so only one file can have its date and time modified by this function. If a device name such as "CON" is specified then no error will be returned but the device's date and time will not actually be changed.

The date and time format are exactly as contained in the directory entry and fileinfo blocks (see chapter 2). No checks are done for sensible dates or times, the values are simply stored. Note that if a fileinfo block is passed then the date and time stored in it will not be updated by this function.

If A=0 then the current values are just returned. Note that although the time value is passed in IX, it is returned in DE. The date and time of a file cannot be altered (although it can be read) if there is a file handle open to the file (".FOPEN" error).

3.3.62 Delete file handle (52h)

- Parameters

```
C = 52H (_HDELETE)
B = File handle
```

- Results

```
A = Error
```

This function deletes the file handle associated with the specified file and closes the file handle. A file handle cannot be deleted if there are any other separately opened file handles open to the same file (".FOPEN" error). If there are any duplicates of the file handle (created by a "duplicate file handle" or "fork" function), then these duplicates will be marked as invalid and any attempt to use them will produce an ".HDEAD" error.

The error conditions for this function are the same as for the "delete file or sub-directory" function (function 4Dh). The file handle will always be closed, even if there is an error condition such as ".FILRO" or ".FOPEN".

3.3.63 Rename file handle (53h)

- Parameters

```
C = 53H (_HRENAME)
B = File handle
HL = New filename ASCIIZ string
```

- Results

```
A = Error
```

This function renames the file associated with the specified file handle with the new name in the string pointed to by HL. Apart from the fact that the file is specified by a file handle rather than an ASCIIZ string or a fileinfo block, this function is identical to the "rename file or subdirectory" function (function 4Eh), and has the same error conditions.

A file handle cannot be renamed if there are any other separately opened file handles for this file (".FOPEN" error), although it can be renamed if there are copies of this file handle, and in this case the copies will be renamed. Renaming a file handle will not alter the file pointer but it will do an implicit "ensure" operation.

3.3.64 Move file handle (54h)

- Parameters

```
C = 54H (_HMOVE)
B = File handle
HL = New path ASCIIZ string
```

- Results

```
A = Error
```

This function moves the file associated with the specified file handle to the directory specified by the new path string pointed to by HL. Apart from the fact that the file is specified by a file handle rather than an ASCIIZ string or a fileinfo block, this function is identical to the "move file or subdirectory" function (function 4Fh), and has the same error conditions.

A file handle cannot be moved if there are any other separately opened file handles for this file (".FOPEN" error), although it can be moved if there are copies of this file handle, and in this case the copies will also be moved. Moving a file handle will not alter the file pointer but it will do an implicit "ensure" operation.

3.3.65 Get/set file handle attributes (55h)

- Parameters

```
C = 55H (_HATTR)
B = File handle
A = 0 => get attributes
    1 => set attributes
L = New attributes byte (only if A=1)
```

- Results

```
A = Error
L = Current attributes byte
```

This function gets or sets the attributes byte of the file associated with the specified file handle. Apart from the fact that the file is specified by a file handle rather than an ASCIIZ string or a fileinfo block, this function is identical to the "get/set file attributes" function (function 50h), and has the same error conditions.

A file handle cannot have its attributes changed (although they can be read) if there are any other separately opened file handles for this file (".FOPEN" error). The file pointer will not be altered but an implicit "ensure" operation will be done.

3.3.66 Get/set file handle date and time (56h)

- Parameters

```
C = 56H (_HFTIME)
B = File handle
A = 0 => get date and time
    1 => set date and time
IX = New time value (only if A=1)
HL = New date value (only if A=1)
```

- Results

```
A = Error
DE = Current file time value
HL = Current file date value
```

This function gets or sets the date and time of the file associated with the specified file handle. Apart from the fact that the file is specified by a file handle rather than an ASCIIZ string or a fileinfo block, this function is identical to the "get/set file date and time" function (function 51h), and has the same error conditions.

A file handle cannot have its date and time changed (although they can be read) if there are any other separately opened file handles for this file (".FOPEN" error). The file pointer will not be altered but an implicit "ensure" operation will be done.

3.3.67 Get disk transfer address (57h)

- Parameters

```
C = 57H (_GETDTA)
```

- Results

```
DE = Current disk transfer address
```

This function returns the current disk transfer address. This address is only used for the *traditional* CP/M style FCB functions and the absolute sector read and write functions.

3.3.68 Get verify flag setting (58h)

- Parameters

```
C = 58H (_GETVFY)
```

- Results

```
B = 00H => verify disabled
    FFH => verify enabled
```

This function simply returns the current state of the verify flag which can be set with MSX-DOS function 2Eh.

3.3.69 Get current directory (59h)

- Parameters

```
C = 59H (_GETCD)
B = Drive number (0=current, 1=A: etc)
DE = Pointer to 64 byte buffer
```

- Results

```
A = Error
DE = Filled in with current path
```

This function simply gets an ASCIIZ string representing the current directory of the specified drive into the buffer pointed to by DE. The string will not include a drive name or a leading or trailing "\" character, so the root directory is represented by a null string. The drive will be accessed to make sure that the current directory actually exists on the current disk, and if not then the current directory will be set back to the root and a null string returned.

3.3.70 Change current directory (5Ah)

- Parameters

```
C = 5AH (_CHDIR)
DE = Drive/path/file ASCIIZ string
```

- Results

```
A = Error
```

The drive/path/file string must specify a directory rather than a file. The current directory of the drive will be changed to be this directory. If the specified directory does not exist then the current setting will be unaltered and a ".NODIR" error returned.

3.3.71 Parse pathname (5Bh)

- Parameters

```
C = 5BH (_PARSE)
B = Volume name flag (bit 4)
DE = ASCIIZ string for parsing
```

- Results

```
A = Error
DE = Pointer to termination character
HL = Pointer to start of last item
B = Parse flags
C = Logical drive number (1=A: etc)
```

This function is purely a string manipulation function, it will not access the disks at all and it will not modify the user's string at all. It is intended to help transient programs in parsing command lines.

The volume name flag (bit 4 of register B; it is in the same bit position as the volume name bit in an attributes byte) determines whether the string will be parsed as a "drive/path/file" string (if the bit is cleared) or a "drive/volume" string (if the bit is set).

The pointer returned in DE will point to the first character which is not valid in a pathname string, and may be the NULL at the end of the string. See chapter 1 for details of the syntax of pathname strings and also for a list of valid characters.

The pointer returned in HL will point to the first character of the last item of a string (filename portion). For example, when a string "A:\XYZ\P.Q /F" was passed, DE will point to the white space character before "/F" and HL will point to "P". If the parsed string ends with a character "\" or is null (apart from drive name), then there will be no "last item", thus HL and DE will point to the same character. In this case, some special procedures will be needed to all the programs which use this function.

The drive number returned in register C is the logical drive specified in the string. If the string did not start with a drive letter then register C will contain the default drive number, since the default drive has been implicitly specified. Register C will never be zero.

The parse flags returned in register B indicate various useful things about the string. For a volume name bits 1, 4, 5, 6 and 7 will always be clear. For a filename, bits 3 to 7 relate to the last item on the string (the "filename" component). The bit assignments are as follows:

b0	Set if any characters parsed other than drive name
b1	Set if any directory path specified
b2	Set if drive name specified
b3	Set if main filename specified in last item
b4	Set if filename extension specified in last item
b5	Set if last item is ambiguous
b6	Set if last item is "." or ".."
b7	Set if last item is ".."

3.3.72 Parse filename (5Ch)

- Parameters

```
C = 5CH (_PFILE)
DE = ASCIIZ string for parsing
HL = Pointer to 11 byte buffer
```

- Results

```
A = Error (always zero)
DE = Pointer to termination character
HL = Preserved, buffer filled in
B = Parse flags
```

This function is purely a string manipulation function, it will not access disks at all and will not modify the string at all. It is intended mainly to help transient programs in printing out filenames in a formatted way. The ASCIIZ string will be parsed as a single filename item, and the filename will be stored in the user's 11 byte buffer in expanded form, with both the filename and the extension padded out with spaces.

The parse flags returned in register B are identical to those for the "parse pathname" function above (function 5Bh), except that bits 0, 1 and 2 will always be clear. The user's buffer will always be filled in, even if there is no valid filename in the string, in which case the buffer will be filled with spaces. "*" characters will be expanded to the appropriate number of "?"s. If either the filename or the filename extension is too long then the excess characters will be ignored.

The pointer returned in register DE will point to the first character in the string which was not part of the filename, which may be the null at the end of the string. This character will never be a valid filename character (see chapter 1 for details of valid filename characters).

3.3.73 Check character (5Dh)

- Parameters

```
C = 5DH (_CHKCHR)
D = Character flags
E = Character to be checked
```

- Results

```
A = 0 (never returns an error)
D = Updated character flags
E = Checked (upper cased) character
```

This function allow language independent upper casing of characters and also helps with handling 16-bit characters and manipulation of filenames. The bit assignments in the character flags are as follows:

b0	Set to suppress upper casing
b1	Set if first byte of 16-bit character
b2	Set if second byte of 16-bit character
b3	Set => volume name (rather than filename)

`b4` Set => not a valid file/volume name character

`b5...b7` Reserved (always clear)

Bit 0 is used to control upper casing. If it is clear then the character will be upper cased according to the language setting of the machine. If this bit is set then the returned character will always be the same as the character passed.

The two 16-bit character flags (bits 1 and 2) can both be clear when the first character of a string is checked and the settings returned can be passed straight back to this function for each subsequent character. Care must be taken with these flags when moving backwards through strings which may contain 16-bit characters.

Bit 4 is set on return if the character is one of the set of filename or volume name terminator characters. Bit 3 is simply used to determine whether to test for filename or volume name characters since the sets are different. 16-bit characters (either byte) are never considered as volume or filename terminators.

3.3.74 Get whole path string (5Eh)

- Parameters

```
C = 5EH (_WPATH)
DE = Pointer to 64 byte buffer
```

- Results

```
A = Error
DE = Filled in with whole path string
HL = Pointer to start of last item
```

This function simply copies an `ASCIIZ` path string from an internal buffer into the user's buffer. The string represents the whole path and filename, from the root directory, of a file or sub-directory located by a previous `"find first entry"` or `"find new entry"` function. The returned string will not include a drive, or an initial `"\"` character. Register `HL` will point at the first character of the last item on the string, exactly as for the `"parse path"` function (function `5Bh`).

If a `"find first entry"` or `"find new entry"` function call is done with `DE` pointing to an `ASCIIZ` string then a subsequent `"get whole path"` function call will return a string representing the sub-directory or file corresponding to the fileinfo block returned by the `"find"` function. If this is a sub-directory then the fileinfo block may be passed back in register `DE` to another `"find first entry"` function call, which will locate a file within this sub-directory. In this case the newly located file will be added onto the already existing whole path string internally, and so a subsequent `"get whole path string"` function call will return a correct whole path string for the located file.

Great care must be taken in using this function because the internal whole path string is modified by many of the function calls, and in many cases can be

invalid. The "get whole path" function call should be done immediately after the "find first entry" or "find new entry" function to which it relates.

3.3.75 Flush disk buffers (5Fh)

- Parameters

```
C = 5FH (_FLUSH)
B = Drive number (0=current, FFH=all)
D = 00H -> Flush only
  = FFH -> Flush and invalidate
```

- Results

```
A = Error
```

This function flushes any dirty disk buffers for the specified drive, or for all drives if B=FFh. If register D is FFh then all buffers for that drive will also be invalidated.

3.3.76 Fork to child process (60h)

- Parameters

```
C = 60H (_FORK)
```

- Results

```
A = Error
B = Process id of parent process
```

This function informs the system that a child process is about to be started. Typically this is a new program or sub-command being executed. For example COMMAND2.COM does a "fork" function call before executing any command or transient program.

A new set of file handles is created, and any current file handles which were opened with the "inheritable" access mode bit set (see the "open file handle" function - function 43h), are copied into the new set of file handles. Any file handles which were opened with the "inheritable" bit clear will not be copied and so will not be available to the child process. The standard file handles (00h...05h) are inheritable and so these will be copied.

A new process id is allocated for the child process and the process id. of the parent process is returned so that a later "join" function call can switch back to the parent process. A ".NORAM" error can be produced by this function if there is insufficient memory to duplicate the file handles.

Because the child process now has a copy of the previous file handles rather than the originals, if one of them is closed then the original will remain open. So for example if the child process closes the standard output file handle (file handle number 1) an re-opens it to a new file, then when a "join" function is done to return to the parent process the original standard output channel will still be there.

3.3.77 Rejoin parent process (61h)

- Parameters

```
C = 61H (_JOIN)
B = Process id of parent, or zero
```

- Results

```
A = Error
B = Primary error code from child
C = Secondary error code from child
```

This function switches back to the specified parent process and returns the error code which the child process terminated with in register B, and a secondary error code from the child in register C. Although the relationship between parent and childprocesses is strictly one-to-one, this function can jump back several levels by giving it a suitable process id. A ".IPROC" error will be returned if the process id is invalid.

The child process's set of file handles are automatically closed and the parent process's set of file handles becomes active again. Also any user RAM segments which the child process had allocated will be freed.

If the process id passed to this function is zero then a partial system re-initialization is done. All file handles are closed and the standard input and output handles re-opened and all user segments are freed. This should not normally be done by a user program if it intends to return to the command interpreter since the command interpreter will not be in a consistent state after this.

This function takes great care that the freeing of memory and adjusting of process id is done before actually closing any file handles and thus before accessing the disk. This ensures that if a disk error occurs and is aborted, the join operation will have been done successfully. However if a "join 0" produces a disk error which is aborted, then the re-initialization of default file handles will not have been done. In this case another "join 0" function call should be done and this will not attempt access disk (because all the files have been closed) and so will be successful.

Note that if this function call is made via 0F37Dh then registers B and C will not return the error codes. This is because program termination and abort handling must be done by the application program. The error code will have been passed to the abort vector and code there must remember the error code if it needs to. See the "terminate with error code" function (function 62h) for the meaning of the primary and secondary error code.

3.3.78 Terminate with error code (62h)

- Parameters

```
C = 62H (_TERM)
B = Error code for termination
```

- Results

```
Does not return
```

This function terminates the program with the specified error code, which may be zero indicating no error. This function call will never return to the caller (unless a user abort routine executes forces it to - see function 63h). The operation of this function is different depending on whether it was called from the MSX-DOS environment via 00005h or from the disk BASIC environment via 0F37Dh.

If called via 00005h then if a user abort routine has been defined by function 63h it will be called with the specified error code (and a zero secondary error code). Assuming that this routine returns, or if there was no user abort routine defined, then control will be passed back to whatever loaded the transient program via a jump at location 00000h. This will almost always be the command interpreter, but in some cases it may be another transient program. The error code will be remembered by the system and the next "join" function (function 61h) which is done will return this error code. The command interpreter will print an error message for any code in the range 20h...FFh, but will not print a message for errors below this.

If this function is called from the disk BASIC environment via 0F37Dh then control will be passed to the abort vector at location "BREAKVECT". In this environment there is no separately defined user abort routine and the error code must be remembered by the code at "BREAKVECT" because "join" will not return the error code.

3.3.79 Define abort exit routine (63h)

- Parameters

```
C = 63H (_DEFAB)
DE = Address of abort exit routine
     0000H to un-define routine
```

- Results

```
A = 0 (never generates errors)
```

This function is only available when called via location 00005h in the MSX-DOS environment. It cannot be called at location 0F37Dh from the disk BASIC environment.

If register DE is zero then a previously defined abort routine will be undefined, otherwise a new one will be defined. The abort routine will be called by the system whenever the transient program is about to terminate for any reason

other than a direct jump to location 0000h. Programs written for MSX-DOS 2 should exit with a "terminate with error code" function call (function 061h) rather than a jump to location 0000h.

The user abort routine will be entered with the user stack active, with IX, IY and the alternate register set as it was when the function call was made and with the whole TPA paged in. The termination error code will be passed to the routine in register A with a secondary error code in register B and if the routine executes a "RET" then the values returned in registers A and B will be stored as the error codes to be returned by the "join" function, and normally printed out by the command interpreter. Alternatively the routine may jump to some warm start code in the transient program rather than returning. The system will be in a perfectly stable state able to accept any function calls.

The primary error code passed to the routine in register A will be the code which the program itself passed to the "terminate with error code" function (which may be zero) if this is the reason for the termination. The routine will also be called if a CTRL-C or CTRL-STOP is detected (".CTRLC" or ".STOP" error), if a disk error is aborted (".ABORT" error), or if an error occurred on one of the standard input or output channels being accessed through MSX-DOS function calls 01h...0Bh (".INERR" or ".OUTERR").

The errors ".ABORT", ".INERR" and ".OUTERR" are generated by the system as a result of some other error. For example a ".ABORT" can result from a ".NRDY" error, or a ".INERR" can result from a ".EOF" error. In these cases the original error code (".NRDY" or ".EOF") is passed to the abort routine in register B as the secondary error code. For all other errors there is no secondary error code and register B will be zero.

If the abort routine executes "POP HL : RET" (or equivalent) rather than a simple return, then control will pass to the instruction immediately following the MSX-DOS call or BIOS call in which the error occurred. This may be useful in conjunction with a disk error handler routine (see function 64h) to allow an option to abort the current MSX-DOS call when a disk error occurs.

3.3.80 Define disk error handler routine (64h)

- Parameters

```
C = 64H (_DEFER)
DE = Address of disk error routine
     0000H to un-define routine
```

- Results

```
A = 0 (never generates errors)
```

This function specifies the address of a user routine which will be called if a disk error occurs. The routine will be entered with the full TPA paged in, but with the system stack in page-3 active and none of the registers will be preserved from when the MSX-DOS function call was made.

The error routine can make MSX-DOS calls but must be very careful to avoid recursion. The list of function calls in section 3.2 of this document indicates which function calls can be safely made from a user error routine. This routine is called with the redirection status being temporarily invalidated in case the standard I/O channels have been redirected. See the "get/set redirection state" function (function 70h) for details of this.

The specification of parameters and results for the routine itself is as below. All registers including IX, IY and the alternate register set may be destroyed but the paging and stack must be preserved. The routine must return to the system, it must not jump away to continue the transient program. If it wants to do this then it should return A=1 ("abort") and a user abort routine will then get control and this may do whatever it wants to.

- Parameters

```
A = Error code which caused error
B = Physical drive
C = b0 - set if writing
    b1 - set if ignore not recommended
    b2 - set if auto-abort suggested
    b3 - set if sector number is valid
DE = Sector number (if b3 of C is set)
```

- Results

```
A = 0 -> Call system error routine
    1 -> Abort
    2 -> Retry
    3 -> Ignore
```

3.3.81 Get previous error code (65h)

- Parameters

```
C = 65H (_ERROR)
```

- Results

```
A = 0
B = Error code from previous function
```

This function allows a user program to find out the error code which caused the previous MSX-DOS function call to fail. It is intended for use with the old CP/M compatible functions which do not return an error code. For example if a "create file FCB" function returns A=0FFh there could be many reasons for the failure and doing this function call will return the appropriate on, for example ".DRFUL" or ".SYSX".

3.3.82 Explain error code (66h)

- Parameters

```
C = 66H (_EXPLAIN)
B = Error code to be explained
DE = Pointer to 64 byte string buffer
```

- Results

```
A = 0
B = 0 or unchanged
DE = Filled in with error message
```

This function allows a user program to get an ASCIIZ explanation string for a particular error code returned by any of the MSX-DOS functions. If an error comes from one of the old functions then "get previous error code" must be called first to get the real error code and then this function can be called to get an explanation string.

Chapter 2 contains a list of all the currently defined error codes and the messages for them. Foreign language versions of the system will of course have different messages. If the error code does have a built in explanation string then this string will be returned and register B will be set to zero. If there is no explanation string then a string of the form: "System error 194" or "User error 45" will be returned, and register B will be unchanged. (System errors are those in the range 40h...FFh and user errors are 00h...3Fh.)

3.3.83 Format a disk (67h)

- Parameters

```
C = 67H (_FORMAT)
B = Drive number (0=current, 1=A:)
A = 00H -> return choice string
    01H...09H -> format this choice
    0AH...FDH -> illegal
    FEH, FFH  -> new boot sector
HL = Pointer to buffer (if A=1...9)
DE = Size of buffer (if A=1...9)
```

- Results

```
A = Error
B = Slot of choice string (only if A=0 on entry)
HL = Address of choice string (only if A=0 on entry)
```

This function is used to format disks and is really only provided for the "FORMAT" command although other programs may use it (with care) if they find it useful.

It has three different options which are selected by the code passed in register A.

If A=0 then registers B and HL return the slot number and address respectively of an ASCIIZ string which specifies the choice of formats which is available. A ".IFORM" error will be returned if this disk cannot be formatted (for example the RAM disk). Normally the string will be read using the "RDSLT" routine and displayed on the screen followed by a "?" prompt. The user then specifies a choice "1"..."9" and this choice is passed back to the "format" function, after a suitable warning prompt, to actually format the disk. If A=0, in some cases zero is returned in HL. This means that there is only one kind of the format and no prompt is required. There is no way of knowing what disk format a particular choice refers to since this is dependant on the particular disk driver.

If A=01h...09h then this is interpreted as a format choice and a disk will be formatted in the specified drive with no further prompting. Register HL and DE must specify a buffer area to be used by the disk driver. There is no way of knowing how big this buffer should be so it is best to make it as big as possible. If the buffer crosses page boundaries then this function will select the largest portion of it which is in one page for passing to the disk driver. Many disk drivers do not use this buffer at all.

If A=FFh then the disk will not actually be formatted, but it will be given a new boot sector to make the disk a true MSX-DOS 2 disk. This is designed to update old MSX-DOS 1.0 disks to have a volume id and thus allow the full disk checking and undeletion which MSX-DOS 2 allows. The case A=FEh is the same as A=FFh except that only the disk parameters are updated correctly and the volume id does not overwrite the boot program. Also there are some MSX-DOS 1.0 implementations which put an incorrect boot sector on the disk and these disks cannot be used by MSX-DOS 2 until they have been corrected by this function.

The "new boot sector" function is mainly intended for the "FIXDISK" utility program, but may be used by other programs if they find it useful. If it is used then a "get format choice" function call (A=0) should be done first and if this returns an error (typically ".IFORM") then the operation should be aborted because this is a drive which does not like to be formatted and the disk could be damaged by this function.

3.3.84 Create or destroy RAMDISK (68h)

- Parameters

```
C = 68H (_RAMD)
B = 00H -> destroy RAM disk
    1...FEH -> create new RAM disk
    FFH -> return RAM disk size
```

- Results

```
A = Error
```

```
B = RAM disk size
```

If register B=0FFh then this routine just returns the number of 16k RAM segments which are allocated to the RAM disk currently. A value of zero indicates that there is no RAM disk currently defined. If B=0 then the current RAM disk will be destroyed, loosing all data which it contained and no error will be returned if there was no RAM disk.

Otherwise, if B is in the range 01h...FEh then this function will attempt to create a new RAM disk using the number of 16k segments specified in register B. An error will be returned if there is already a RAM disk (".RAMDX") or if there is not even one segment free (".NORAM"). If there are insufficient free RAM segments to make a RAM disk of the specified size then the largest one possible will be created. No error is returned in this case.

In all cases the size of the RAM disk will be returned in register B as a number of segments. Note that some of the RAM is used for the file allocation tables and the root directory so the size of the RAM disk as indicated by "DIR" or "CHKDSK" will be somewhat smaller than the total amount of RAM used. The RAM will always be assigned the drive letter "H:" regardless of the number of drives in the system.

3.3.85 Allocate sector buffers (69h)

- Parameters

```
C = 69H (_BUFFER)
B = 0 -> return number of buffers
         else number of buffers required
```

- Results

```
A = Error
B = Current number of buffers
```

If B=0 then this function just returns the number of sector buffers which are currently allocated. If B!=0 then this function will attempt to use this number of sector buffers (must always be at least 2). If it cannot allocate as many as requested then it will allocate as many as possible and return the number in register B but will not return an error. The number of sector buffers can be reduced as well as increased.

The sector buffers are allocated in a 16k RAM segment outside the normal 64k so the number of buffers does not detract from the size of the TPA. However the number of buffers does affect efficiency since with more buffers allow more FAT and directory sectors to be kept resident. The maximum number of buffers will be about 20.

3.3.86 Logical drive assignment (6Ah)

- Parameters

```
C = 6AH (_ASSIGN)
B = Logical drive number (1=A: etc)
D = Physical drive number (1=A: etc)
```

- Results

```
A = Error
D = Physical drive number (1=A: etc)
```

This function controls the logical to physical drive assignment facility. It is primarily intended for the "ASSIGN" command although user programs may want to use it to translate logical drive numbers to physical drive numbers.

If both B and D are non-zero then a new assignment will be set up. If register B is non-zero and register D is zero then any assignment for the logical drive specified by B will be cancelled. If both register B and D are zero then all assignments will be cancelled. If register B is non-zero and register D is FFh then the current assignment for the logical drive specified by register B will simply be returned in register D.

All drives used in the various function calls, including drive names in strings and drive numbers as parameters to function calls, are logical drives. However the drive number passed to disk error routines is a physical drive so if "ASSIGN" has been used these may be different from the corresponding logical drive.

3.3.87 Get environment item (6Bh)

- Parameters

```
C  = 6BH (_GENV)
HL = ASCIIZ name string
DE = Pointer to buffer for value
B  = Size of buffer
```

- Results

```
A  = Error
DE = Preserved, buffer filled in if A=0
```

This function gets the current value of the environment item whose name is passed in register HL. A ".IENV" error is returned if the name string is invalid. If there is no environment item of that name then a null string will be returned in the buffer. If there is an item of that name then its value string will be copied to the buffer. If the buffer is too small then the value string will be truncated with no terminating null and a ".ELONG" error will be returned. A buffer 255 bytes will always be large enough since value strings cannot be longer than this (including the terminating NULL).

3.3.88 Set environment item (6Ch)

- Parameters

```
C  = 6CH (_SENV)
HL = ASCIIZ name string
DE = ASCIIZ value string
```

- Results

```
A = Error
```

This function sets a new environment item. If the name string is invalid then a ".IENV" error is returned, otherwise the value string is checked and a ".ELONG" error returned if it is longer than 255 characters, or a ".NORAM" error if there is insufficient memory to store the new item. If all is well then any old item of this name is deleted and the new item is added to the beginning of the environment list. If the value string is null then the environment item will be removed.

3.3.89 Find environment item (6Dh)

- Parameters

```
C  = 6DH (_FENV)
DE = Environment item number
HL = Pointer to buffer for name string
```

- Results

```
A  = Error
HL = Preserved, buffer filled in
```

This function is used to find out what environment items are currently set. The item number in register DE identifies which item in the list is to be found (the first item corresponds to DE=1). If there is an item number <DE> then the name string of this item will be copied into the buffer pointed to by HL. If the buffer is too small then the name will be truncated with no terminating null, and a ".ELONG" error returned. A 255 byte buffer will never be too small. If there is no item number <DE> then a null string will be returned, since an item can never have a null name string.

3.3.90 Get/set disk check status (6Eh)

- Parameters

```
C = 6EH (_DSKCHK)
A = 00H -> get disk check status
    01H -> set disk check status
B = 00H -> enable (only if A = 01H)
    FFH -> disable (only if A = 01H)
```

- Results

```
A = Error
B = Current disk check setting
```

If A=0 then the current value of the disk check variable is returned in register B. If A=01h then the variable is set to the value in register B. A value of 00h means that disk checking is enabled and a non-zero means that it is disabled. The default state is enabled.

The disk check variable controls whether the system will re-check the boot sector of a disk to see whether it has changed, each time a file handle, fileinfo block or FCB is accessed. If it is enabled then it will be impossible to accidentally access the wrong disk by changing a disk in the middle of an operation, otherwise this will be possible and may result in a corrupted disk. Depending on the type of disk interface, there may be some additional overhead in having this feature enabled although with many types of disk (those with explicit disk change detection hardware) it will make no difference and the additional security is well worth having.

3.3.91 Get MSX-DOS version number (6Fh)

- Parameters

```
C = 6FH (_DOSVER)
```

- Results

```
A  = Error (always zero)
BC = MSX-DOS kernel version
DE = MSXDOS2.SYS version number
```

This function allows a program to determine which version of MSX-DOS it is running under. Two version numbers are returned, one in BC for the MSX-DOS kernel in ROM and the other is DE for the MSXDOS2.SYS system file. Both of these version numbers are BCD values with the major version number in the high byte and the two digit version number in the low byte. For example if there were a version 2.34 of the system, it would be represented as 0234h.

For compatibility with MSX-DOS 1.0, the following procedure should always be followed in using this function. Firstly if there is any error (A!=0) then it is not MSX-DOS at all. Next look at register B. If this is less than 2 then the system is earlier than 2.00 and registers C and DE are undefined. If register B is 2 or greater then registers BC and DE can be used as described above. In general the version number which should be checked (after this procedure) is the MSXDOS2.SYS version in register DE.

3.3.92 Get/set redirection state (70h)

- Parameters

```
C = 70H (_REDIR)
A = 00H - get redirection state
    01H - set redirection state
B = New state: b0 - standard input
               b1 - standard output
```

- Results

```
A = Error
B = Redirection state before command:
      b0 set - input is redirected
      b1 set - output is redirected
```

This function is provided primarily for disk error routines and other character I/O which must always go to the console regardless of any redirection. When the CP/M character functions (functions `01h...0Bh`) are used, they normally refer to the console. However if the standard input or output file handles (file handles 0 and 1) have been closed and reopened to a disk file, then the CP/M character functions will also go to the disk file. However certain output such as disk error output must always go to the screen regardless.

This function allows any such redirection to be temporarily cancelled by calling this function with `A`=1 and `B`=0. This will ensure that any subsequent CP/M console I/O will go to the console, and will also return the previous setting so that this can be restored afterwards. The system is a somewhat unstable state when the redirection state has been altered like this and there are many function calls which will reset the redirection to its real state over-riding this function. In general any function call which manipulates file handles, such as `"open"`, `"close"`, `"duplicate"` and so on, will reset the redirection state. The effect of this function is therefore purely temporary.