# 8086 Programmer's Guide

First Edition: January 1984

## ABOUT THIS GUIDE

**Purpose**

We've designed this 8086 Programmer's Guide to provide the information you need to know in order to write application software to run on 8086-family microcomputers under the TurboDOS operating system. This document explains the theory of operation of each internal facility of TurboDOS. It also describes in detail each TurboDOS function that may be called by an application program.

**Assumptions**

In writing this guide, we've assumed that you are an experienced assembly-language programmer writing application programs for the 8086 TurboDOS environment. We've also assumed you have read the TurboDOS 1.3 User's Guide, and are therefore familiar with the commands and external features of TurboDOS.

**Organization**

This guide starts with a section that describes the fundamentals of the TurboDOS environment, with emphasis on the organization of memory and the interface and flow of control between application programs and the operating system.

The next two sections explain TurboDOS internals in more detail. One describes the file system, and the other describes serial I/O.

There are two reference sections that explain each TurboDOS function call in detail. One section describes CP/M-compatible functions supported by TurboDOS, while the other describes functions unique to TurboDOS.

Appendices describe the TurboDOS 8086 assembler, linker, and debugger. The document concludes with a summary of function calls, and an alphabetical index.

**Related Documents**

In addition to this guide, you might be interested in four other related documents:

- TurboDOS 1.3 User's Guide
- TurboDOS 1.3 8086 Implementor's Guide
- TurboDOS 1.3 Z80 Programmer's Guide
- TurboDOS 1.3 Z80 Implementor's Guide

You should read the User's Guide before you start into this document. It introduces the external features and facilities of TurboDOS, and describes each TurboDOS command in detail.

You'll need the 8086 Implementor's Guide if you are adapting TurboDOS to a new hardware configuration. It explains the system generation and OEM distribution procedures, and also describes how to implement hardware-dependent driver modules.

You'll need the Z80 guides if you are programming or configuring a TurboDOS system that uses Z80 microprocessors.

C-FUNCTIONS (Continued)



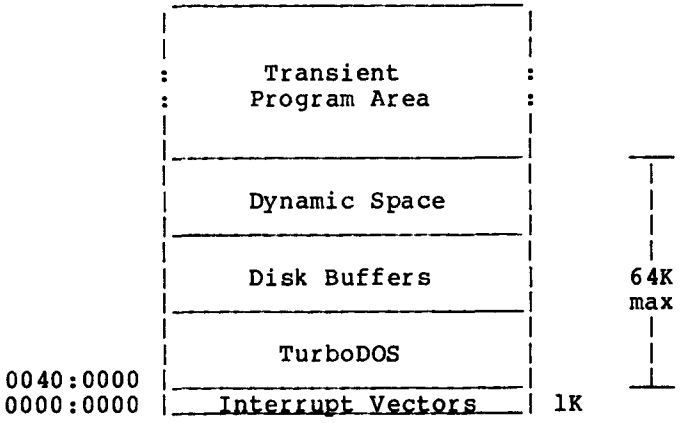
T-FUNCTIONS

## FUNDAMENTALS

This section introduces you to the TurboDOS environment. Emphasis is given to the organization of memory, and to the interface and flow of control between application programs and the operating system. Subsequent sections describe the file system and other facilities in detail.

## Memory Organization

The resident portion of TurboDOS may be anywhere in the one-megabyte address space supported by an 8086-family CPU. Usually, it is loaded at location 0040:0000 hex, immediately above the lower 1K reserved by the 8086 architecture for interrupt vectors. Immediately following the TurboDOS resident is an area of memory reserved for disk buffers and other dynamic working storage. The remaining memory space available for use by commands and application programs is known as the "Transient Program Area" (TPA).



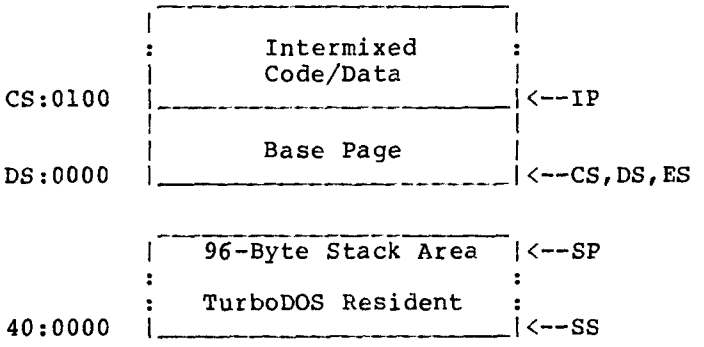

Under 8086 TurboDOS, several transient programs may be loaded into the TPA at one time (although only one may be in execution).

## Execution Models

Transient programs are stored in files of type .CMD, preceded by a header record which defines the segmentation and memory allocation requirements of the program. Transient programs may be written as a single group with intermixed code and data ("8080 Model"), with separate code and data groups ("Small Model"), or with up to eight separate groups: code, data, extra, stack, and up to four auxilliary groups ("Compact Model").

### 8080 Model

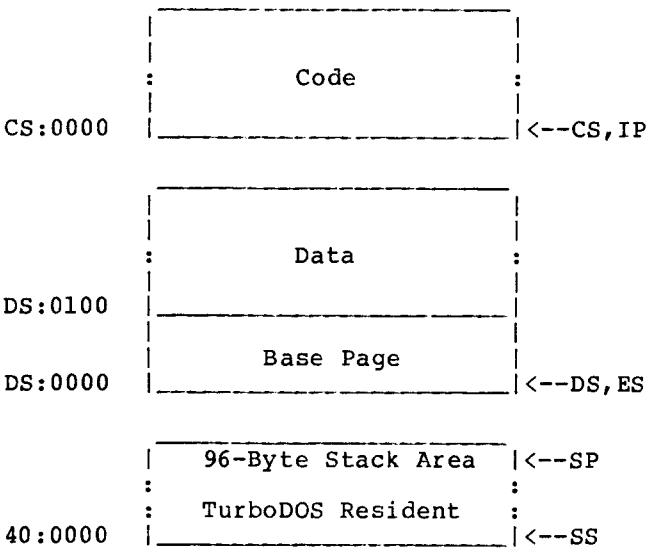
If the .CMD header defines only a code group, then it is assumed that the code and data portions of the program are intermixed. TurboDOS allocates a TPA segment sufficient to contain the code group. The first 256 bytes of the code group is assumed to be a Base Page reserved for communications between the operating system and the program.




For this "8080 Model", TurboDOS initializes the CS, DS, and ES segment registers to address the single code group. The IP register is set to 0x0100 so that execution starts immediately following the Base Page. The SS and SP registers initially point to a 96-byte stack area provided within TurboDOS.

Small Model

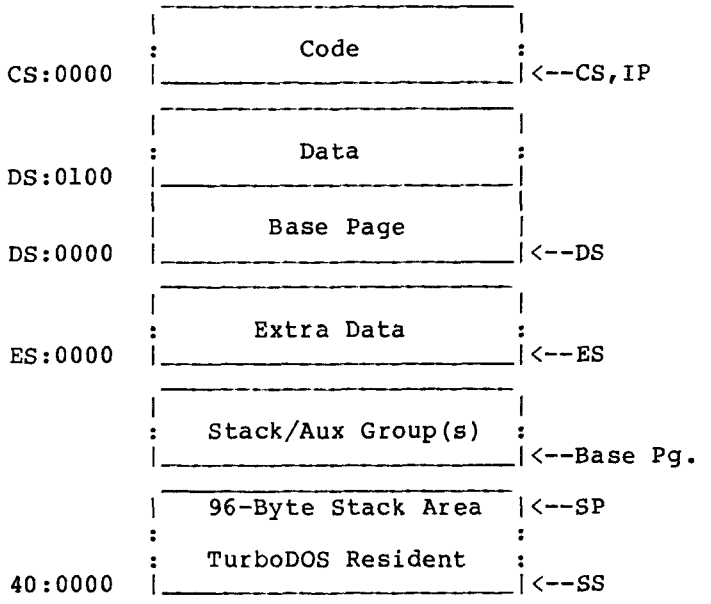
If the .CMD header defines both a code group and a data group, then it is assumed that the code and data portions of the program are separate and independent. In this case, TurboDOS allocates separate TPA segments for the code group and the data group. The two allocated segments are not necessarily contiguous. The Base Page is assumed to occupy the first 256 bytes of the data group.




For this "Small Model", TurboDOS initializes the CS register to the base of the code group, and initializes the DS and ES registers to address the base of the data group. The IP register is set to zero. The SS and SP registers initially point to a 96-byte stack area provided within TurboDOS.

Compact Model

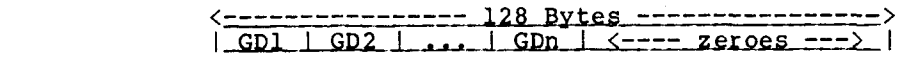
If the .CMD header defines a code group, a data group, and one or more additional groups (extra, stack, or auxilliary), then TurboDOS allocates separate TPA segments (not necessarily contiguous) for each of the groups. The Base Page is assumed to occupy the first 256 bytes of the data group.




For this "Compact Model", TurboDOS initializes the CS and DS registers to the base of the code and data groups, respectively. ES is set to the base of the extra group if present, otherwise to the data group. The IP register is set to zero. The SS and SP registers initially point to a 96-byte stack area provided within TurboDOS. The stack and auxilliary groups may be located via pointers in the Base Page.

## Command Files

A transient command file (type .CMD) always starts with a 128-byte header record that defines the segment structure and allocation requirements of the transient program. The header record contains from one to eight "group descriptors", each nine bytes long. The balance of the 128 bytes is zero-filled.




Each 9-byte group descriptor has this format:

| G-Type (byte) | G-Size (word) | G-Abs (wrd) | G-Min (word) | G-Max (word) |
|---|---|---|---|---|

The G-Type field designates the group type:

| G-Type | Group Type |
|---|---|
| 1 | Code Group |
| 2 | Data Group |
| 3 | Extra Group |
| 4 | Stack Group |
| 5 | Aux-1 Group |
| 6 | Aux-2 Group |
| 7 | Aux-3 Group |
| 8 | Aux-4 Group |

The G-Size field specifies the number of paragraphs of loadable memory-image data to be read from the .CMD file for this group.

The G-Abs field is ignored by TurboDOS, and normally set to 0x0000.

The G-Min and G-Max fields specify the minimum and maximum number of paragraphs to be allocated for this group.

Following the header record, the command file contains the loadable portion of each group in memory-image format, in the same order as the group descriptors in the header record.

**Program Interface**

TurboDOS supports 103 different functions that may be invoked by an application program. Functions are provided for file management, memory management, console input/output, printing and spooling, and various other TurboDOS facilities. The last half of this guide is largely devoted to describing each of these functions in detail.

Functions supported by TurboDOS fall into two categories: CP/M-compatible functions, and TurboDOS-unique functions. We will refer to them as "C-functions" and "T-functions", respectively. TurboDOS supports 60 C-functions and 43 T-Functions.

C-Functions

To invoke a C-function, a program executes an interrupt instruction INT 224 (or INT 0xE0) with a function number in the CL-register. TurboDOS supports all CP/M-86 BDOS functions:

| | |
|---|---|
| 0 System Reset | 20 Read Sequential |
| 1 Console Input | 21 Write Sequential |
| 2 Console Output | 22 Make File |
| 3*Raw Console Input | 23 Rename File |
| 4*Raw Console Output | 24 Return Login Vector |
| 5 List Output | 25 Return Current Disk |
| 6 Direct Console I/O | 26 Set DMA Address |
| 7 Get I/O Byte | 27*Get ALV Address |
| 8 Set I/O Byte | 28 Write Protect Disk |
| 9 Print String | 29 Get R/O Vector |
| 10 Read Cons. Buffer | 30 Set File Attributes |
| 11 Get Console Status | 31 Get DPB Address |
| 12 Return Version | 32 Get/Set User Number |
| 13 Reset Disk System | 33 Read Random |
| 14 Select Disk | 34 Write Random |
| 15 Open File | 35 Compute File Size |
| 16 Close File | 36 Set Random Record |
| 17 Search for First | 37 Reset Drive |
| 18 Search for Next | (38-39 reserved) |
| 19 Delete File | 40*Write Random 0-Fill |

C-Functions (Continued)

| | | | |
|---|---|---|---|
| 50 | Direct BIOS Call | 55 | Allocate Memory |
| 51 | Set DMA Base | 56 | Allocate Abs Memory |
| 52 | Get DMA Base | 57 | Free Memory |
| 53 | Alloc Max Memory | 58 | Free All Memory |
| 54 | Alloc Abs Max Mem | 59 | Program Load |

These TurboDOS C-functions are compatible with the corresponding functions in CP/M-86 except for the four functions marked with an asterisk above. In TurboDOS, C-functions 3 and 4 are compatible with MP/M-86 rather than CP/M-86. C-function 40 is synonymous with 34. C-function 27 (Get ALV Address) performs no operation in TurboDOS, but this function affects only the STAT utility of CP/M-86 which is not normally used with TurboDOS.

In addition to BDOS functions 0-40 and 50-59 supported by CP/M-86, a number of additional functions have been implemented in Concurrent CP/M-86 and MP/M-86. TurboDOS provides compatible C-functions for certain of these functions:

| | | | |
|---|---|---|---|
| 42 | Lock Record | 107 | Return Serial No. |
| 43 | Unlock Record | 108 | Get/Set Rtn Code |
| 46 | Get Free Space | 110 | Get/Set Delimiter |
| 47 | Chain to Program | 111 | Print Block |
| 104 | Set Date/Time | 112 | List Block |
| 105 | Get Date/Time | 152 | Parse Filename |

However, the following rarely-used functions are <u>not</u> implemented, and perform no function in 8086 TurboDOS:

| | | | |
|---|---|---|---|
| 41 | Test and Write | 99 | Truncate File |
| 44 | Set Multi-Sector | 100 | Set Dir Label |
| 45 | Set Error Mode | 101 | Get Dir Label |
| 48 | Flush Buffers | 102 | Read PW Mode |
| 49 | Get/Set SCB | 103 | Write File XFCB |
| 60 | Call RSX | 106 | Set Default PW |
| 98 | Free Blocks | 109 | Get/Set Cons Mode |

T-Functions

To invoke a T-function, a program executes an interrupt instruction INT 225 (or INT 0xE1) with a function number in the CL-register. A different entrypoint interrupt is used to avoid conflict with C-function numbers. TurboDOS supports the following T-functions:

| | | | |
|---|---|---|---|
| 0 | Reset O/S | 22 | Phys Disk Access |
| 1 | Create Process | 23 | Set Buffer Parms |
| 2 | Delay Process | 24 | Get Buffer Parms |
| 3 | Allocate Memory | 25 | Lock/Unlock Drive |
| 4 | Deallocate Memory | 26 | Flush/Free Buffers |
| 5 | Send Message | 27 | Get/Set Print Mode |
| 6 | Receive Message | 28 | Sig End-of-Print |
| 7 | Set Error Address | 29 | Get/Set Despl Mode |
| 8 | Set Abort Address | 30 | Queue a Print File |
| 9 | Set Date/Time | 31 | Flush List Buffer |
| 10 | Get Date/Time | 32 | Network List Out |
| 11 | Rebuild Disk Map | 33 | Remote Console I/O |
| 12 | Get TurboDOS S/N | 34 | Get Comm Status |
| 13 | Set Compat. Flags | 35 | Comm Input |
| 14 | Log-On/Log-Off | 36 | Comm Output |
| 15 | Load File | 37 | Set Comm Baud Rate |
| 16 | Activate Do-File | 38 | Get Comm Baud Rate |
| 17 | Autoload On/Off | 39 | Set Modem Controls |
| 18 | Send Command Line | 40 | Get Modem Status |
| 19 | Get Alloc Info | 41 | User-Defined Func. |
| 20 | Get Phys Disk Info | 42 | Reorg Disk Directory |
| 21 | Get/Set Drv Status | | |

Termination

A program may terminate by invoking C-function 0 (System Reset), or alternatively by executing a far-return instruction "RETF" (provided the original values of the SS and SP registers are intact). Both methods are entirely equivalent, and cause TurboDOS to terminate the program in TPA and prompt for the next command. A program may also terminate by invoking C-function 47 (Chain to Program), which allows the program to specify the next command to be executed after the program terminates.

**Command Processing**

A TurboDOS command always identifies a program file residing on disk, and causes that program to be loaded into memory (TPA) and executed. TurboDOS has no "built-in" commands.

TurboDOS comes with more than 30 standard command programs (described in detail in the User's Guide). You can expand the vocabulary of commands simply by storing additional programs on disk. Programs are usually kept in .CMD files.

Command Prompt

TurboDOS displays a command prompt on the console whenever it is ready to accept a command. The command prompt is composed of the current user number, the current drive letter, and the } prompt symbol.

Command Format

Each TurboDOS command consists of the file name of the program to be executed, possibly followed by an optional command tail of up to 126 characters. A command may be entered in upper- or lower-case letters, but is converted to upper-case by TurboDOS.

The program name may have an explicit file type, but usually doesn't (TurboDOS assumes .CMD). It may also have an explicit drive specification (like "B:") if the program is not on the current drive. You will get an error message if the program file cannot be found on disk, or if the available TPA is not big enough to hold the program.

A special kind of command is used to change the current drive. It consists of a drive specification (like "B:") with no program name.

---

Tail Parsing

The format of a command tail is determined by the particular program involved. TurboDOS passes the command tail to the program by saving the length of the tail (in characters) at location DS:0080 of the Base Page, and saving the text of the tail (up to 126 characters) starting at location DS:0081. TurboDOS also stores a null (zero byte) immediately following the last character of the command tail. The tail includes all characters following the program name, including leading spaces. If no tail is given in the command, the length stored at DS:0080 is zero.

If the command tail consists of one or two filenames of the form:

{d:}filename{.typ}

then TurboDOS parses each into File Control Block (FCB) format. The first parsed FCB is saved at location DS:005C of the Base Page, and the second parsed FCB is saved at location DS:006C. Parsing is done following the procedure described for C-function 152 (Parse Filename).

---

Command Strings

TurboDOS also accepts strings of commands separated by the character \ (backslant). TurboDOS executes each command in sequence, and re-displays each but the first as it is executed. A command string may not exceed the size of the command buffer, which is normally big enough to accomodate two lines of text.

Batch Processing

TurboDOS supports a batch processing mode in which execution is controlled by a predefined sequence of commands stored in a "do-file" on disk. A do-file is a text file (usually type .DO), each line of which contains a valid TurboDOS command or command string. A do-file may be activated with a DO command, or by invoking T-function 16 (Activate Do-File). A do-file may contain any number of embedded DO commands, and nesting is supported to any reasonable depth.

Automatic Loading

TurboDOS provides a facility for loading any program or executing any command sequence automatically at initial start-up (cold start) or whenever a program terminates (warm start). Autoload at cold-start takes place only if a file named COLDSTRT.AUT is present on the start-up disk. Autoload at warm-start takes place only if a file named WRMxSTRT.AUT (where x=8 for UP8s, 6 for UP16s, and B for the background batch) is present on the current disk. The AUTOLOAD command is the usual way to create these .AUT files.

Alternatively, a program (.CMD file) may be autoloaded by renaming it as COLDSTRT.AUT or WRM6STRT.AUT. In this case, however, the autoloaded program must not rely on the contents of the Base Page FCB (at DS:005C) and buffer (at DS:0080), because they will be left uninitialized after the autoload.

## Base Page Layout

The Base Page is the 256-byte memory region from DS:0000 to DS:00FF. The Base Page is initialized by TurboDOS whenever a transient program is loaded, and is used for communication between TurboDOS and the transient program. The organization of the Base Page is shown below:

| Hex Addr | Description |
|---|---|
| 0000-0002 | Length of code group in bytes. Stored as a 24-bit number, least-significant byte first. |
| 0003-0004 | Base paragraph address of code group. |
| 0005 | 8080 Model flag, set to 1 if 8080 Model, 0 otherwise. |
| 0006-0008 | Length of data group in bytes, 24 bits, LSB first. |
| 0009-000A | Base paragraph address of data group. |
| 000B | (Unused, reserved.) |
| 000C-000E | Length of extra group in bytes, 24 bits, LSB first. |
| 000F-0010 | Base paragraph address of extra group. |
| 0011 | (Unused, reserved.) |
| 0012-0014 | Length of stack group in bytes, 24 bits, LSB first. |
| 0015-0016 | Base paragraph address of stack group. |

Base Page Layout (Continued)

| Hex Addr | Description |
|---|---|
| 0017 | (Unused, reserved.) |
| 0018-001A | Length of aux-1 group in bytes, 24 bits, LSB first. |
| 001B-001C | Base paragraph address of aux-1 group. |
| 001D | (Unused, reserved.) |
| 001E-0020 | Length of aux-2 group in bytes, 24 bits, LSB first. |
| 0021-0022 | Base paragraph address of aux-2 group. |
| 0023 | (Unused, reserved.) |
| 0024-0026 | Length of aux-3 group in bytes, 24 bits, LSB first. |
| 0027-0028 | Base paragraph address of aux-3 group. |
| 0029 | (Unused, reserved.) |
| 002A-002C | Length of aux-4 group in bytes, 24 bits, LSB first. |
| 002D-002E | Base paragraph address of aux-4 group. |
| 002F-005B | (Unused, reserved.) |
| 005C-006B | Default FCB part 1. The first filename argument in a command tail is parsed into this 16-byte area. |

**Base Page Layout (Continued)**

| Hex Addr | Description |
|---|---|
| 006C-007B | Default FCB part 2. The second filename argument in a command tail is parsed into this 16-byte area, and must be moved to another location before making use of the default FCB. |
| 007C | Default FCB current record. |
| 007D-007F | Default FCB random record. |
| 0080-00FF | Default 128-byte buffer. This area receives the command tail length in 0080H, and the command tail text (up to 126 characters plus a null terminator) in locations 0081H-00FFH. |

## System Start-Up

To get TurboDOS started, it is necessary to read a copy of the operating system from disk into memory, a process known as "cold start". The exact cold-start procedure depends on the particular hardware involved.

Most TurboDOS implementations use this three-step cold-start procedure:

1. When the computer is turned on or reset, it executes the TurboDOS bootstrap from read-only memory (ROM). (In some implementations, the bootstrap may be loaded from reserved tracks on disk.) The bootstrap scans all disk drives from A to P, searching the directory of each ready drive for a file named OSLOAD.COM which contains the TurboDOS loader. When this file is found, the bootstrap loads it into the TPA and executes it.

2. The TurboDOS loader scans all disk drives from A to P, searching for a file named OSSERVER.SYS which contains the server operating system. When this file is found, the loader proceeds to load the operating system into memory, then transfers control to it. The drive from which the OSSERVER.SYS file was loaded becomes the "system disk".

3. The server downloads a user bootstrap routine into each user processor. The server then locates a file named OSUSER .SYS on the system disk which contains the user operating system, and downloads it into each user processor.

During network operation, it is helpful if the system disk is always on-line. If a fixed disk is available, it should be used as the system disk.

## Summary

This section has introduced the fundamentals of the TurboDOS environment. You have learned how memory is organized, how programs may be segmented into various execution models, and how .CMD files are formatted. You understand the TurboDOS program interface, including C-functions, T-functions, and direct BIOS calls. You know how TurboDOS parses and processes commands, command strings, and do-files, and how it communicates with programs via the Base Page.

Next, we examine the TurboDOS file system in considerable detail.

## FILE SYSTEM

This section describes the TurboDOS file system in detail. It covers the structure of disks and files, the facilities provided to manage files, and the procedures for calling these facilities from application programs.

### Disk Capacity

The TurboDOS file system can support up to sixteen logical drives per processor, identified by the letters A through P. Drives may be local to the processor, or may be attached to another processor and accessed by means of networking.

TurboDOS accomodates any combination of drives from mini-floppies to large hard disks in excess of a gigabyte. Allocation block size may be chosen individually for each drive, and affects maximum drive capacity as follows:

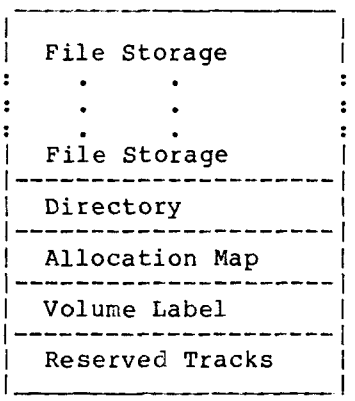
| Alloc. Block Size | Max. Drive Capacity |
|---|---|
| 1K | 256 Kilobytes |
| 2K | 128 Megabytes |
| 4K | 256 Megabytes |
| 8K | 512 Megabytes |
| 16K | 1,024 Megabytes |

Because these limits are so big, it is almost never necessary to partition a physical drive into smaller logical drives under TurboDOS. However, such partitioning is sometimes done for user convenience when using large fixed disks.

For maximum capacity and performance, floppy disks used with TurboDOS are generally formatted with large sector sizes (512 or 1024 bytes), no interleave, and no reserved tracks. However, TurboDOS also accomodates standard CP/M floppy disk formats.

**Disk Organization**

Each disk is organized into five areas:




Reserved tracks are required by certain hardware configurations to support cold-start, but are not otherwise used by TurboDOS. The volume label permits a name to be given to each disk. The allocation map contains one bit for each allocation block on the disk, and is used by TurboDOS to keep track of which disk blocks are occupied and which are free. The directory is a table of contents which identifies all files stored on the disk. The remainder of the disk (most of it) is available for file storage.

CP/M does not maintain a volume label or allocation map on the disks it creates. When a CP/M disk is first accessed by TurboDOS, the first few CP/M directory entries are automatically relocated to the end of the directory in order to make room for the label and map. When a TurboDOS disk is accessed by CP/M, the label and map appear to be ordinary deleted directory entries. Thus, disks can be moved freely between CP/M and TurboDOS in spite of the differences in organization.

## Directory Formats

TurboDOS supports two alternative directory formats: linear and hashed. A flag bit in the directory label indicates which format is in use on a particular disk.

The standard linear format is compatible with CP/M, and is searched sequentially. Consequently, look-up speed deteriorates with increasing directory size, and can get painfully slow on large disks with many files.

The optional hashed directory format uses a hashing algorithm to make look-up in large directories much faster. A hashed directory may be used on any disk, but is especially suited for use on hard disks with many files. Hashed directories are not media-compatible with CP/M, but may be converted to linear format whenever exporting to CP/M is needed.

Whether the directory is linear or hashed, searches involving "wild cards" have to be done linearly. Such wild-card searches are typically slower if the directory is hashed.

## File Organization

A file contains a sequence of 128-byte records, and may be up to 134 megabytes (1,048,576 records) long. The records of a file may be read and written sequentially or randomly (by relative record number). A file may be extended by writing beyond the end of file. TurboDOS automatically allocates disk space when a file is extended, and deallocates it when a file is deleted.

Text files are written as a sequence of ASCII characters with a carriage-return (0x0D) and line-feed (0x0A) at the end of each text line. Text lines are variable length and may span records. The end of a text file is marked by the ASCII character SUB (0x1A).

## File Operations

About half of the 60 C-functions supported by TurboDOS are connected with the file system. These functions support the operations needed to manipulate files, directories, and disks.

The following functions provide the basic facilities for sequential file access:

| C-Fcn | Function Name |
|---|---|
| 15 | Open File |
| 16 | Close File |
| 20 | Read Sequential |
| 21 | Write Sequential |
| 22 | Make File |

These additional functions are necessary to support random access and file sharing:

| C-Fcn | Function Name |
|---|---|
| 33 | Read Random |
| 34 | Write Random |
| 35 | Compute File Size |
| 36 | Set Random Record |
| 42 | Lock Record |
| 43 | Unlock Record |

Directory functions include:

| C-Fcn | Function Name |
|---|---|
| 17 | Search for First |
| 18 | Search for Next |
| 19 | Delete File |
| 23 | Rename File |
| 30 | Set File Attributes |

**File Operations (Continued)**

Drive-oriented functions are:

| C-Fcn | Function Name |
|---|---|
| 14 | Select Disk |
| 24 | Return Login Vector |
| 25 | Return Current Disk |
| 28 | Write Protect Disk |
| 29 | Get R/O Vector |
| 31 | Get DPB Address |
| 37 | Reset Drive |
| 46 | Get Free Space |

Finally, some other functions connected with the file system include:

| C-Fcn | Function Name |
|---|---|
| 13 | Reset Disk System |
| 26 | Set DMA Address |
| 32 | Set/Get User Number |
| 47 | Chain to Program |
| 51 | Set DMA Base |
| 52 | Get DMA Address |
| 59 | Program Load |
| 152 | Parse Filename |

Each of these file system C-functions is described in detail later in this document.

## Naming Files

TurboDOS keeps track of files by name, maintaining a directory of files on each disk. A file is identified uniquely by four fields:

- drive letter (A-P)
- user number (0-31)
- file name (up to 8 characters)
- file type (up to 3 characters)

The drive letter specifies the disk on which the file is located. If no drive letter is given, the current drive is assumed by default.

The user number specifies one of 32 logical file libraries on each disk. These libraries allow files to be conveniently segregated by user or application. Generally, user 0 is reserved for global files and user 31 is reserved for log-on security, leaving 1-30 for general use.

The name and type fields are composed of ASCII characters. The file name may have up to eight characters, and the file type may have up to three. Shorter names and types are padded on the right with spaces.

It is suggested that file names and file types be composed from the upper-case letters A-Z and the digits 0-9. Actually, any ASCII characters may be used including lower-case letters, punctuation, and even non-printing control characters. However, such names may not be parsed correctly in commands nor displayed correctly in directories.

The question mark ? is a special wild-card character which may be used in file names and types to match any character in the corresponding position during directory searches.

**Special File Names**

TurboDOS gives special meaning to two reserved file names. "$.DIR" refers to the directory area of a disk, while "$.DSK" refers to the entire contents of the physical disk volume (up to the maximum file size of 134 megabytes). These special files may be dumped, patched, or accessed like any ordinary file. However, access is restricted to privileged log-ons only.

**File Control Block**

File-oriented C-functions and T-functions are always called with the address of a File Control Block (FCB) in the DX-register. The FCB is a data structure 33 bytes long (36 bytes for random access operations) organized as follows:

| Offset | Field | Description |
|---|---|---|
| 0 | drive | drive code (0-16):<br>0 -> current drive<br>1 -> drive A<br>2 -> drive B<br>:<br>16 -> drive P |
| 1-8 | name | file name in ASCII, padded on right with spaces, high-order bit of each byte reserved for attributes f1-f8 |
| 9-11 | type | file type in ASCII, padded on right with spaces, high-order bit of each byte reserved for attributes t1-t3 |
| 12 | extent | least significant five bits of extent number |

**File Control Block (Continued)**

| Offset | Field | Description |
|---|---|---|
| 13 | specl | flag byte (Do Not Use) |
| 14 | spec2 | most significant eight bits of extent number |
| 15 | record count | number of records in current extent (0-128) |
| 16-31 | map | allocation map of current extent |
| 32 | current record | current record number (0-127) in current extent |
| 33-35 | random record | 20-bit record number (byte 33 is least significant) for random-access operations |

In general, the application program must initialize FCB bytes 0-12 before opening, making, or searching for a file. It must also zero FCB byte 32 before reading or writing a file sequentially from the beginning.

When a file is opened, TurboDOS fills FCB bytes 0-31 with information from the directory. Thereafter, the application program should not modify FCB bytes 0-31. When the file is closed, TurboDOS updates the directory with information from the FCB. A directory entry has the same structure as the first 32 bytes of an FCB. In a directory entry, however, byte 0 contains the user number 0-31 to which the file belongs, or the value 0xE5 if the directory entry is not in use. Also, byte 13 may contain the exact byte count of the last record in the file.

**File Attributes**

File attributes are stored in the high-order bits of the FCB name field bytes f1-f8 and type field bytes t1-t3, and are used to control how a file may be accessed:

| Attribute | Definition |
|---|---|
| f1 | FIFO file attribute |
| f2-f4 | undefined file attributes |
| f5-f8 | interface attributes |
| t1 | read-only file attribute |
| t2 | global file attribute |
| t3 | archived attribute |

The file attribute bits f1-f4 and t1-t3 are recorded in the directory, and may be set or cleared by means of C-function 30 (Set File Attributes). For a newly-created file, all attribute bits are initialized to zero. When a file is opened, its attributes are copied into the FCB. File attributes may also be interrogated by means of C-functions 17 and 18 (Search for First/Next).

The read-only attribute (t1) prevents a file from being written, deleted or renamed. The global attribute (t2) enables a file saved under user 0 to be accessed from any user number (it has no effect for files saved under non-zero user numbers). The archived attribute (t3) is used for incremental file backup, and is automatically cleared by TurboDOS whenever a file is written or renamed. The FIFO attribute (f1) causes a file to be accessed using a special "first-in first-out" access method (described later).

Attributes f2-f4 are undefined, and available to the user. Interface attribute bits f5-f8 cannot be used as file attributes; they specify options for certain C-functions.

## User Numbers

TurboDOS provides 32 file libraries on each disk corresponding to user numbers 0-31. Generally, user 0 is reserved for global files and user 31 is reserved for log-on security, leaving 1-30 for general use.

The current user number is established initially at log-on. For a non-privileged log-on, the user number remains unchanged until log-off. This restricts file access to the corresponding file library (plus global files under user 0). For a privileged log-on, the user number may be changed without restriction by means of C-function 32 (Set/Get User Number).

The current user number is treated as a prefix to file names, thereby allowing each disk directory to contain up to 32 libraries. Most directory functions (make, rename, delete, search, etc.) are restricted to the library corresponding to the current user number. However, files in the user 0 library which have the global file attribute may be opened from any user number. This permits commands, programs, and other common files to be shared by all users.

## File Sharing

In a multi-user TurboDOS system, it is possible for multiple users to access the same file at the same time. This can happen if the users are logged-on to the same user number, or accessing the same global file. TurboDOS supports interlocks to regulate such file sharing at the file or record level.

TurboDOS file sharing facilities are compatible with MP/M, but provide significant extensions to alleviate the most serious deficiencies in MP/M file sharing.

File Locks

File-level interlocks are supported by means of four distinct modes of opening a file. The open mode is determined by FCB interface attributes f5-f6 when the file is opened or created. The four open modes are called exclusive, shared, read-only, and permissive.

A file opened in exclusive mode is available to the opening process exclusively until it is closed, and may not be opened by any other process. A file cannot be opened in exclusive mode if the file is currently opened (in any mode) by another process.

A file may be opened in shared mode by any number of processes simultaneously. All processes are allowed to read, write and extend the file. Record lock and unlock functions are honored only for file opened in shared mode.

A file may be opened in read-only mode by any number of processes simultaneously. All processes are allowed to read the file, but not to write or extend it.

A file may be opened in permissive mode by any number of processes simultaneously. All processes are allowed to read the file. If any process writes or extends the file, then that process gains an exclusive write-lock on the file, preventing any other process from writing to the file. The exclusive write-lock is released when the locking process closes the file.

In shared and permissive modes, if a process extends a file by adding new records at the end, these records become immediately accessible to other processes that also have the file open.

Record Locks

Record-level interlocks are controlled by means of explicit locking and unlocking requests made by the application program. This allows concurrent update by multiple processes.

Record locks are by no means automatic, and require explicit cooperative participation by all updating programs. C-functions 42 (Lock Record) and 43 (Unlock Record) are honored only for files opened in the shared mode. Each program must lock a record before reading it, and must unlock the record after updating it.

If a program attempts to lock a record that is already locked by another process, the Lock Record function returns an error code and the program must try again until it is successful. Alternatively, the program can ask TurboDOS to suspend program execution automatically until the lock request can be satisfied.

To extend a shared file in a concurrent update environment, the extending program should first acquire a lock on record N+1 (where N is the last record in the file). The program may then safely write record N+1, and finally unlock N+1.

Compatibility Modes

The file sharing facilities of TurboDOS are designed to provide compatibility with MP/M, yet at the same time to alleviate the most serious limitations of MP/M file sharing. TurboDOS may be instructed to adhere strictly to MP/M file-sharing rules, or alternatively to relax some of these rules. To this end, TurboDOS provides a byte of "compatibility flags" with the following bit assignments:

| Bit | Flag Name | Affects |
|---|---|---|
| 7 | permissive | default open mode |
| 6 | suspend | lock conflict action |
| 5 | global-write | writing global files |
| 4 | mixed-mode | mixed file open modes |
| 3 | logical | record lock validity |
| 2-0 | (not defined) | |

For each compatibility flag, a zero-bit denotes strict adherence to the MP/M rule, while a one-bit signifies a relaxation of that rule. The initial setting of the compatibility flags may be established during TurboDOS system generation by assigning the desired value to the symbolic location COMPAT. A program may modify its compatibility flags by calling T-function 13 (Set Compatibility Flags), but the flags automatically revert to their initial setting when the program terminates.

If the permissive flag (bit 7) is set, the default file open mode is permissive, rather than exclusive (as in MP/M). Specifically, the open mode is determined when a file is opened or created by FCB interface attributes f5-f6, as shown in the following table:

Compatibility Modes (Continued)

| permissive flag = 0 | | | permissive flag = 1 | | |
|---|---|---|---|---|---|
| f6 | f5 | open mode | f6 | f5 | open mode |
| 0 | 0 | exclusive | 0 | 0 | permissive |
| 0 | 1 | shared | 0 | 1 | shared |
| 1 | 0 | read-only | 1 | 0 | read-only |
| 1 | 1 | permissive | 1 | 1 | exclusive |

If the suspend flag (bit 6) is set, then an attempt to lock a record that is already locked by someone else causes the process to be suspended until its lock request can be satisfied. Otherwise, an attempt to lock or write to a record that is already locked by someone else results in an immediate error return code (as in MP/M).

If the global-write flag (bit 5) is set, then a program running under a non-zero user number may both read and write global files. Otherwise, access to global files is strictly read-only (as in MP/M).

If the mixed-mode flag (bit 4) is set, then one process may open a file in shared mode while another has it open in read-only mode (or vice-versa). Otherwise, the shared and read-only modes are mutually exclusive (as in MP/M).

If the logical flag (bit 3) is set, then the FCB random record field for C-functions 42 and 43 (Lock/Unlock Record) is interpreted as an arbitrary 24-bit logical record number which is not validated and does not cause file positioning. Otherwise, the FCB random record field for C-functions 42 and 43 is interpreted as the relative number of a 128-byte record, and causes the file to be positioned to that record (as in MP/M).

## FIFO Files

To facilitate communications between processes, processors and users, TurboDOS supports a special kind of file called a FIFO (first-in, first-out) similar in concept to a Unix pipe. FIFOs are opened, closed, read and written exactly like ordinary sequential files. However, a record written to a FIFO is always appended to the end, and a record read from a FIFO is always taken from the beginning and removed from the FIFO.

A FIFO is differentiated from other files by the presence of the FIFO attribute (f1) in the directory. Record zero of a FIFO is a header record used by TurboDOS to keep track of the FIFO, and is organized as follows:

| Offset | Contents |
|---|---|
| 0 | type (0=RAM, -1=disk) |
| 1 | mode (0=error code, -1=suspend) |
| 2-3 | maximum size (records) |
| 4-5 | current size (records) |
| 6-7 | number of last record read |
| 8-9 | number of last record written |
| 10-127 | (not used, reserved) |

The header specifies whether the body of the FIFO is RAM- or disk-resident, and the maximum number of records it may contain. RAM-resident FIFOs provide high-speed but limited capacity (up to 127 records, usually much less). Disk-resident FIFOs provide large capacity (up to 65,535 records) but slower speed. The FIFO command may be used to create a FIFO and initialize its header.

**FIFO Files (Continued)**

Normally, reading from an empty FIFO returns an end-of-file code (A=1), and writing to a full FIFO returns a disk-full code (A=2). However, if the mode byte in the FIFO header is set to -1 (suspend), then reading from an empty FIFO or writing to a full FIFO causes the process to be suspended until the FIFO becomes non-empty or non-full.

The header or disk-resident body of a FIFO may be accessed directly using C-functions 33 and 34 (Read/Write Random), thereby bypassing the normal first-in first-out protocol. An attempt to make (C-function 22) an existing FIFO is treated as an open (C-function 15), while an attempt to delete (C-function 19) a FIFO is ignored. The only way to get rid of a FIFO is first to clear the FIFO attribute, then delete it.

## Buffer Management

The TurboDOS buffer manager performs multi-level buffering of physical disk input/output, using least-recently-used (LRU) buffer assignment and other sophisticated optimizations. Buffering provides a manyfold reduction in the number of physical disk accesses during both sequential and random file operations.

The number and/or size of disk buffers may be changed by means of T-function 23 (Set Buffer Parameters), and interrogated by T-function 24 (Get Buffer Parameters). The number of buffers must be at least two, and the buffer size must be at least as large as the physical sector size of the disks being used. For optimum performance, the number of buffers should be as large as possible consistent with the TPA size required.

The buffer manager maintains its buffers on two lists: the "in-use" list and the "free" list. Whenever the file manager requests a disk access, the buffer manager first checks the in-use list to see if the requested disk sector is already in a buffer. Most of the time it is, and no physical disk access is required. If not, the buffer manager attempts to acquire a new buffer from the free list. If the free list is empty, the least-recently-used buffer (at the end of the in-use list) is written out to disk if necessary, and then reused to receive the newly requested disk sector.

**Media Changes**

Before a removable disk volume is changed, it is crucial that any buffers relating to that disk are written out if necessary, and returned to the free list. In single-user configurations of TurboDOS, this is done automatically whenever the system pauses for console input. In multi-user configurations, buffers must be explicitly flushed and freed by calling T-function 26 (Flush/Free Buffers) prior to changing disks. This is most commonly done by executing the CHANGE command, but should also be coded into applications that require media changes during operation. For safety, TurboDOS also flushes buffers automatically during any lull in system activity, and frees them automatically whenever a disk drive becomes not-ready.

**Error Handling**

In the event of an unrecoverable disk error, TurboDOS normally displays a diagnostic message in one of these formats:

```
Read Error, Drive A, Track 0, Sector 2
[Retry, Ignore, Abort]

Write Error, Drive B, Track 5, Sector 16
[Retry, Ignore, Abort]

Not Ready Error, Drive C [Retry, Abort]

Spooler Error [Ignore, Abort]
```

and waits for the user to choose the desired recovery option by keying in the appropriate letter (R, I or A).

**Error Handling (Continued)**

An application program may elect to intercept and process such errors, however, by calling T-function 7 (Set Error Address). In this case, TurboDOS does not display its usual diagnostic messages. Normal error processing resumes automatically when the application program terminates.

NOTE: Because the buffer manager optimizes disk write operations by deferring them as long as possible, write errors may be reported later than expected and possibly even to a different user than expected.

**Error Handling (Continued)**

An application program may elect to intercept and process such errors, however, by calling T-function 7 (Set Error Address). In this case, TurboDOS does not display its usual diagnostic messages. Normal error processing resumes automatically when the application program terminates.

NOTE: Because the buffer manager optimizes disk write operations by deferring them as long as possible, write errors may be reported later than expected and possibly even to a different user than expected.

## SERIAL I/O

This section describes the TurboDOS facilities that deal with serial input/output (I/O) in connection with consoles, printers, and communications channels.

## Console I/O

TurboDOS provides ten C-functions that permit programs to interact with the user console device. Three kinds of console input/output are supported in TurboDOS: basic I/O, raw I/O, and string I/O.

### Basic Console I/O

Three C-functions provide basic console I/O on a single-character basis:

| C-Fcn | Function Name |
|---|---|
| 1 | Console Input |
| 2 | Console Output |
| 11 | Get Console Status |

The Console Input function waits for a character to be keyed in, echoes the character to the console screen to provide visual confirmation, and returns the character to the calling program.

The Console Output function displays a character on the console screen. It expands horizontal tab characters into spaces, based upon tab stops at every eighth column.

The Get Console Status function checks to see whether or not a console input character is available, and returns a Boolean result.

Raw Console I/O

Three additional C-functions provide raw console I/O:

| C-Fcn | Function Name |
|---|---|
| 3 | Raw Console Input |
| 4 | Raw Console Output |
| 6 | Direct Console I/O |

The Raw Console Input function is similar to the basic Console Input function, except that input characters are not echoed to the screen. Likewise, the Raw Console Output function is like the basic Console Output function, except that horizontal tabs are not expanded.

The Direct Console I/O function combines the functions of Raw Console Input, Raw Console Output, and Console Status. It is supported only for compatibility with CP/M.

String Console I/O

The remaining console I/O functions provide input and output of character strings:

| C-Fcn | Function Name |
|---|---|
| 9 | Print String |
| 10 | Read Console Buffer |
| 110 | Get/Set Delimiter |
| 111 | Print Block |

The Print String function outputs a string of characters to the console. The string may be of any length, and is terminated by a reserved delimiter. The delimiter is normally the dollar-sign $ character, but may be changed by means of the Get/Set Delimiter function.

String Console I/O (Continued)

The Print Block function is similar to Print String, except that the string length is passed explicitly so that no delimiter is needed. Both Print String and Print Block expand horizontal tabs.

The Read Console Buffer function reads an entire line of edited input from the console. Characters are accepted from the console and stored in successive memory locations until a carriage-return terminates the line. Input characters are echoed to console output (but, unlike CP/M, tabs are not expanded). Rudimentary editing is supported: backspace or delete characters erase the last typed character, while CTRL-U or CTRL-X erase the entire line.

Attention Requests

The execution of a program or do-file may be suspended at any time by typing a reserved "attention" character on the console keyboard. In most installations, this is either CTRL-S or BREAK. TurboDOS will "beep" to acknowledge that it has received the attention request.

After an attention request, the interrupted program or do-file will remain suspended until one of the following attention responses is typed:

CTRL-Q (resume) simply restarts execution at the point of interruption.

CTRL-C (abort) cancels execution of the interrupted program or do-file, causes any nested commands and do-files to be disregarded, and returns to the command prompt. An application program may elect to intercept such abort requests, however, by calling T-function 8 (Set Abort Address).

Attention Requests (Continued)

CTRL-P (echo-print) restarts execution and causes all subsequent console output also to be echoed to the printer. A second attention/echo sequence turns off echoing of console output to the printer.

CTRL-L (end-print) restarts execution after signalling the end of the current print job.

**Comm Channel I/O**

In order to allow communications-oriented applications programs to be written in a hardware-independent fashion, TurboDOS supports a standard communications channel interface consisting of seven T-functions:

| T-Fcn | Function Name |
|---|---|
| 34 | Get Comm Channel Status |
| 35 | Comm Channel Input |
| 36 | Comm Channel Output |
| 37 | Set Comm Channel Baud Rate |
| 38 | Get Comm Channel Baud Rate |
| 39 | Set Comm Channel Modem Controls |
| 40 | Get Comm Channel Modem Status |

These functions support multiple channels of communications. T-functions 34-36 provide basic single-character comm channel I/O (analogous to raw console I/O). T-functions 37-38 allow programs to sense or set the comm channel baud rate to any standard speed from 50 to 19,200 baud. T-functions 39-40 allow programs to set modem control signals (RTS, DTR) and to sense modem status signals (CTS, DSR, DCD, RI).

## Printer Output

TurboDOS provides the basic printing functions of CP/M, plus an elaborate concurrent printing facility which offers several modes of print spooling and flexible print routing among multiple printers and print queues. The spooling and routing facilities are completely transparent to application programs.

### Basic Printing

Two C-functions provide the basic means for programs to generate printer output:

| C-Fcn | Function Name |
|---|---|
| 5 | List Output |
| 112 | List Block |

The List Output function outputs a single character to be printed, while the List Block function outputs a character string of specified length. In contrast to console I/O, these print output functions do not expand tabs.

### Control Functions

Four T-functions provide control over the print spooling, de-spooling, and queuing mechanisms of TurboDOS:

| T-Fcn | Function Name |
|---|---|
| 27 | Get/Set Print Mode |
| 28 | Signal End-of-Print |
| 29 | Get/Set De-Spool Mode |
| 30 | Queue a Print File |

The Get/Set Print Mode function controls print routing. Print output may be routed direct to a specified printer, spooled to a specified drive and print queue, displayed on the console, or simply discarded.

Control Functions (Continued)

The Signal End-of-Print function allows a program to terminate a print job explicitly. In the absence of this function, a print job ends automatically at the conclusion of the program, upon receipt of an end-print attention request from the console, or when a reserved end-of-print character (if defined) appears in the print output stream.

The Get/Set De-Spool Mode function controls background printing (de-spooling). A printer may be assigned to de-spool from a specified queue, or may be placed in an off-line status. Any print job in process may be stopped, resumed, restarted from the beginning, or terminated altogether.

The Queue a Print File function permits a program to queue a print file (or any text file, for that matter) for background printing. The file may be placed on any specified print queue, and may be saved or deleted automatically after printing.

## C-FUNCTIONS

This section describes the 60 CP/M-compatible functions ("C-functions") supported by TurboDOS. The C-functions are presented in numerical order, with calling parameters, return value, and a detailed explanation for each.

To invoke a C-function, a program executes an interrupt instruction INT 224 (or INT 0xE0) with a function number in register CL. Byte-length arguments are passed in register DL, and word-length arguments in register DX. In the case of a memory location argument, the segment base is passed in DS and the offset in DX.

C-functions return byte-length values in register AL (duplicated in BL), or word-length values in register BX (duplicated in AX). A few functions return memory location values in ES (base) and BX (offset).

If a C-function call is made with register CL set to an unsupported function number, TurboDOS returns immediately with registers BX and AX zeroed.

C-function calls generally destroy registers AX-BX-CX-DX-SI-DI-BP-ES but preserve SP-IP and CS-DS-SS.

---

**C-Function 0** **System Reset**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 0 | |

Explanation

The System Reset function terminates the calling program ("warm-start"). Program termination also may be accomplished by executing a far return instruction RETF (provided the original values of registers SS and SP have been preserved) and has exactly the same effect.

In a multi-user TurboDOS system, program termination closes any open files, releases any locked records or devices, and ends any active print job.

---

**C-Function 1** **Console Input**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 1 | |

Returned Value

| Reg | Description |
|---|---|
| AL = input character | |

Explanation

The Console Input function obtains the next character from the console keyboard, and returns it in register AL. If no character is available, the calling program is suspended until a character is typed.

Graphic characters and certain control characters (carriage-return, line-feed, and backspace) are echoed to the console screen. Horizontal tabs are expanded into multiple spaces, based upon tab stops at every eighth column.

---

**C-Function 2** **Console Output**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 2 | |
| DL = output character | |

Explanation

The Console Output function displays the character passed in register DL on the console screen. Horizontal tabs are expanded into multiple spaces, based upon tab stops at every eighth column.

**C-Function 3** **Raw Console Input**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 3 | |

Returned Value

| Reg | Description |
|---|---|
| AL = input character | |

Explanation

The Raw Console Input function obtains the next character from the console keyboard, and returns it in register AL. If no character is available, the calling program is suspended until a character is typed. Input characters are not echoed to the console screen.

This function is compatible with MP/M-86. (In CP/M-86, this function is Input from Reader Device. In Concurrent CP/M, this function is Auxiliary Input.)

---

**C-Function 4** **Raw Console Output**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 4 | |
| DL = output character | |

Explanation

The Raw Console Output function displays the character passed in register DL on the console screen. Horizontal tabs are not expanded.

This function is compatible with MP/M-86. (In CP/M-86, this function is Output to Punch Device. In Concurrent CP/M, this function is Auxiliary Output.)

---

**C-Function 5** **List Output**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 5 | |
| DL = output character | |

Explanation

The List Output function sends the character passed in register DL to be printed according to the current print routing. Horizontal tabs are not expanded.

---

**C-Function 6** **Direct Console I/O**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 6 | |
| DL = -1 | (for combined status/input) |
| | -2 (for status) |
| | -3 (for raw input) |
| | output character (for raw output) |

Returned Value

| Reg | Description |
|---|---|
| AL = | input character or status |

Explanation

The Direct Console I/O function performs one of four possible sub-functions, depending upon the argument passed in register DL.

If DL = -1 (0xFF), then any available console input character is returned in register AL, without echo to the screen. If no character is available, the function returns AL = 0.

If DL = -2 (0xFE), then this function returns console status (A = 0 if no console input is available, or AL = -1 otherwise). Equivalent to C-function 11 (Get Console Status).

If DL = -3 (0xFD), then this function obtains the next console input character and returns it in register AL, without echo to the screen. If no character is available, the calling program is suspended until a character is typed. Equivalent to C-function 3 (Raw Console Input).

For other values of DL, this function displays the character on the console screen. Horizontal tabs are not expanded. Equivalent to C-function 4 (Raw Console Output).

**C-Function 6 (Continued)**

Note that the 8086 TurboDOS implementation of this function is compatible with MP/M-86, Concurrent CP/M, CP/M-80, and Z80 TurboDOS. It differs somewhat from the implementation in CP/M-86, however.

---

**C-Function 7** **Get I/O Byte**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 7 | |

Returned Value

| Reg | Description |
|---|---|
| AL = contents of I/O byte | |

Explanation

This function simply returns the value of the memory location identified by the public name IOBYTE# (used in some implementations to control serial I/O device assignment).

NOTE: This function is supported and IOBYTE# is defined only if the optional module CPMSUP is included during TurboDOS system generation.

---

**C-Function 8** **Set I/O Byte**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 7 | |
| DL = new value of I/O byte | |

Explanation

This function simply sets the value of the memory location identified by the public name IOBYTE# (used in some implementations to control serial I/O device assignment).

NOTE: This function is supported and IOBYTE# is defined only if the optional module CPMSUP is included during TurboDOS system generation.

---

**C-Function 9** **Print String**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 9 | |
| DS:DX = string address | |

Explanation

The Print String function displays a string of characters on the console screen. The string may be of any length, and is terminated by a reserved delimiter. The delimiter is normally the dollar-sign $ character, but may be changed by means of C-function 110 (Get/Set Output Delimiter). Horizontal tabs are expanded into multiple spaces, based upon tab stops at every eighth column.

**C-Function 10** **Read Console Buffer**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 10 | |
| DS:DX = buffer address | |

Explanation

The Read Console Buffer function reads an entire line of edited input from the console. The input buffer whose address is passed in registers DS:DX has the following structure:

| Offset | Direction | Description |
|---|---|---|
| 0 | passed | max input size (N) |
| 1 | returned | actual input (0-N) |
| 2 to N+1 | returned | input characters |

The first byte of the buffer must be preset to the maximum number of characters allowed in the input line.

Console input is accepted until terminated by a carriage-return. Input errors may be corrected by typing BACKSPACE or DELETE to erase one character at a time, or CTRL-U or CTRL-X to erase the entire line. Characters in excess of the maximum are not accepted, and diagnosed with a "beep". Input characters are echoed to the console screen. Unlike CP/M, this function does not expand tabs in TurboDOS.

Upon return, the second byte of the buffer contains the actual number of input characters in the buffer. The input line is returned starting at the third byte of the buffer. The terminating carriage-return is neither stored in the buffer nor included in the count. Unused buffer positions following the last input character are uninitialized.

---

**C-Function 11** **Get Console Status**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 11 | |

Returned Value

| Reg | Description |
|---|---|
| AL = -1 if console input is available<br>0 if console input is not available | |

Explanation

The Get Console Status function checks to see whether or not a console input character is available. If console input is available, it returns AL = -1, otherwise it returns AL = 0.

---

**C-Function 12** **Return Version**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 12 | |

Returned Values

| Reg | Description |
|---|---|
| BH = 0x00 (meaning: CP/M, not MP/M) | |
| BL = 0x31 (meaning: BDOS version 3.1) | |

Explanation

The Return Version function provides information on the latest compatible version of CP/M. (The BDOS version number returned in register BL may be changed by patching the symbol CPMVER during system generation.)

---

**C-Function 13** **Reset Disk System**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 13 | |

Explanation

In TurboDOS, the only effect of the Reset Disk System function is to reset the current DMA offset to 0x0080. (See C-function 26, Set DMA Offset.)

**C-Function 14** **Select Disk**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 14 | |
| DL = selected disk drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |

Explanation

The Select Disk function causes the disk drive specified in register DL to be selected as the current (default) disk drive. The current drive is used in subsequent file operations whenever the FCB drive field is set to zero.

---

**C-Function 15** **Open File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 15 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if file not found | |

Explanation

The Open File function opens the file specified by the FCB drive, name, type, and extent fields (bytes 0 through 12). Normally, the extent field (byte 12) should be set to zero. The specified file must exist under the current user number or must be a global file under user 0.

The open mode is determined by compatibility flag bit 7 (permissive) and by the FCB interface attributes f5 and f6, as shown in the following table:

| permissive flag = 0 | | | permissive flag = 1 | | |
|---|---|---|---|---|---|
| f6 | f5 | open mode | f6 | f5 | open mode |
| 0 | 0 | exclusive | 0 | 0 | permissive |
| 0 | 1 | shared | 0 | 1 | shared |
| 1 | 0 | read-only | 1 | 0 | read-only |
| 1 | 1 | permissive | 1 | 1 | exclusive |

If the FCB current record field (byte 32) is set to -1, this function returns the byte count of the last record of the file in the current record field. The calling program should zero the current record field before doing sequential reads or writes.

**C-Function 16** **Close File**

Entry Arguments

| Reg | Description |
|---|---|
| CL | = 16 |
| DS:DX | = FCB address |

Returned Value

| Reg | Description |
|---|---|
| AL | = 0 if successful<br>-1 if file not found |

Explanation

The Close File function closes a file previously opened by an Open File (15) or Make File (22) C-function. The directory is updated if necessary to reflect any new blocks allocated to the file, and any locked records are unlocked.

If FCB interface attribute f5 is set, this function performs a "partial close" operation which updates the directory but leaves the file open.

**C-Function 17** **Search for First**

Entry Arguments

| Reg | Description |
|---|---|
| CL | = 17 |
| DS:DX | = FCB address |

Returned Value

| Reg | Description |
|---|---|
| AL | = entry number (0-3) if successful<br>-1 if file not found |

Explanation

The Search for First function scans the directory for the first entry which matches the FCB drive, name, type, and extent fields (bytes 0 through 12) and the current user number. An ASCII question mark (0x3F) in any FCB byte 1 through 12 is treated as a wild-card which matches any character in the corresponding byte position of the directory entry.

If the search is successful, this function returns a directory record (containing four 32-byte directory entries) at the current DMA address, and a value in register AL (0-3) that indicates which of the four entries was found to match the FCB. If the search is not successful, the function returns -1 (0xFF) in register AL.

If the Search for First function succeeds in finding an entry which matches the given FCB, then C-function 18 (Search for Next) may be called repeatedly to locate all remaining matches in the directory.

Explanation (Continued)

A special situation occurs if the FCB drive field (byte 0) is set to a question mark (0x3F). In this case, the remainder of the FCB is ignored, the directory of the current drive is searched, and the Search for First function returns the very first directory entry (usually the volume label). The Search for Next function will then return each successive directory entry in sequence, regardless of user number. Even deleted entries are returned in this case.

**C-Function 18** **Search for Next**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 18 | |

Returned Value

| Reg | Description |
|---|---|
| AL = entry number (0-3) if successful<br>-1 if file not found | |

Explanation

The Search for Next function continues the search initiated by C-function 17 (Search for First). If the search is successful, this function returns a directory record (containing four 32-byte directory entries) at the current DMA address, and with a value in register AL (0-3) that indicates which of the four entries was found to match the FCB. If the search is not successful, the function returns -1 (0xFF) in register AL.

---

**C-Function 19** **Delete File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 19 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if no file was deleted | |

Explanation

The Delete File function deletes the file specified by the FCB drive, name, and type fields (bytes 0 through 11) and the current user number. ASCII question marks (0x3F) may be used as wild-cards anywhere in the FCB name and type fields, in which case this function deletes all matching files.

A program may delete a file that it has open, in which case a close is performed implicitly before the file is deleted. However, a program is not permitted to delete a file that another process has open, nor a file that has the read-only or FIFO attributes.

If FCB interface attribute f5 is set, this function performs no operation and returns AL=0 to indicate successful completion. (This is for compatibility with M/PM and Concurrent CP/M, where the f5 attribute causes only XFCBs to be deleted.)

**C-Function 20** **Read Sequential**

Entry Arguments

| Reg | Description |
|---|---|
| CL | = 20 |
| DS:DX | = FCB address |

Returned Value

| Reg | Description |
|---|---|
| AL | = 0 if successful |
| | 1 if at end-of-file |
| | 128 if FCB current record invalid |

Explanation

The Read Sequential function reads the next 128-byte record from a file into memory at the current DMA address. The given FCB must have been previously opened by an Open (15) or Make (22) C-function, and the FCB current record field (byte 32) initialized to zero.

This function uses the FCB extent and current record fields to determine the record to be read, then increments the current record field in preparation for the next sequential operation. If the current record field overflows, the next extent is opened and the current record field is reset to zero.

**C-Function 21** **Write Sequential**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 21 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 1 | if file too large (>134 Mb) |
| 2 | if disk full or file read-only |
| 8 | if attempt to write locked record |
| 128 | if FCB current record invalid |
| -1 | if no directory space |

Explanation

The Write Sequential function writes the next 128-byte record of a file from the current DMA address in memory. The given FCB must have been previously opened by an Open (15) or Make (22) C-function, and the FCB current record field (byte 32) initialized to zero.

This function uses the FCB extent and current record fields to determine the record to be write, then increments the current record field in preparation for the next sequential operation. If the current record field overflows, the next extent is opened (or created if it does not exist) and the current record field is reset to zero.

**C-Function 22** **Make File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 22 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if directory full, file exists, or FCB invalid | |

Explanation

The Make File function creates a new (empty) file specified by the FCB drive, name, type, and extent fields (bytes 0 through 12). Normally, the extent field (byte 12) should be set to zero. The directory entry for the new file is placed under the current user number. All file attributes are initialized to zero. A request to make a file that already exists is denied.

The newly-created file is left in an open state. If the FCB interface attribute f5 is set, then the file is left open in shared mode. Otherwise, the file is left open in either exclusive or permissive mode, depending on compatibility flag bit 7 (permissive).

The calling program should zero the FCB current record field (byte 32) before doing sequential reads or writes on the file.

**C-Function 23** **Rename File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 23 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if file not found, file in-use, or file name invalid | |

Explanation

The Rename File function renames the file specified by the FCB drive, name, and type fields (bytes 0 through 11) and the current user number. The file is given the new name and type specified in bytes 17 through 27 of the FCB. Wild-card characters (ASCII question marks) are not allowed in either the old or new name. All remaining bytes of the FCB are disregarded by this function.

A program may rename a file that it has open, in which case a close is performed implicitly before the file is renamed. However, a program is not permitted to rename a file that another process has open, nor a file that has the read-only attribute.

**C-Function 24** **Return Login Vector**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 24 | |

Returned Value

| Reg | Description |
|---|---|
| BX = login vector | |

Explanation

The Return Login Vector function tests the ready status of all disk drives. It returns a 16-bit vector in register BX containing a one-bit for each drive that is ready for access, and a zero-bit for each drive that is not ready or not defined. The least significant bit corresponds to drive A, and the most significant bit to drive P.

NOTE: This function is supported only if the optional module CPMSUP is included during TurboDOS system generation.

**C-Function 25** **Return Current Disk**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 25 | |

Returned Value

| Reg | Description |
|---|---|
| AL = current disk drive:<br>0 for drive A<br>1 for drive B<br>:<br>15 for drive P | |

Explanation

The Return Current Disk function returns the identity of the current (default) disk drive in register AL.

---

**C-Function 26** **Set DMA Offset**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 26 | |
| DX = DMA offset address | |

Explanation

The Set DMA Offset function causes the offset address specified in register DX to be used as the record buffer address for subsequent file read and write operations. The DMA offset is relative to the current DMA base (see C-function 51).

Whenever a program is loaded into the TPA, the DMA base is initialized to the data segment base of the program. The DMA offset is initialized to 0x0080, the address of the default record buffer in the Base Page. C-function 13 (Reset Disk System) also sets the DMA offset to 0x0080.

---

**C-Function 27** **Get ALV Address**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 27 | |

Returned Value

| Reg | Description |
|---|---|
| BX = 0 | |

Explanation

This function performs no operation in TurboDOS. (Under CP/M, it returns the address of the memory-resident allocation vector for the current disk.)

---

**C-Function 28** **Write Protect Disk**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 28 | |

Explanation

The Write Protect Disk function marks the current (default) disk drive as read-only, preventing any program from writing to the disk. C-function 37 (Reset Drive) must be used to enable writes to the disk once again.

Unlike CP/M, TurboDOS does not re-enable writing after warm-start, C-function 0 (System Reset), or C-function 13 (Reset Disk System). Consequently, write-protection of a disk drive is not nearly so temporary as it is in CP/M.

NOTE: This function is supported only if the optional module CPMSUP is included during TurboDOS system generation.

---

**C-Function 29** **Get Read-Only Vector**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 29 | |

Returned Value

| Reg | Description |
|---|---|
| BX = read-only vector | |

Explanation

The Get Read-Only Vector function returns a 16-bit vector in register BX containing a one-bit for each disk drive that is write-protected, and a zero-bit for each drive that is not. The least significant bit corresponds to drive A, and the most significant bit to drive P.

NOTE: This function is supported only if the optional module CPMSUP is included during TurboDOS system generation.

**C-Function 30** **Set File Attributes**

Entry Arguments

| Reg | Description |
|---|---|
| CL | = 30 |
| DS:DX | = FCB address |

Returned Value

| Reg | Description |
|---|---|
| AL | = 0 if successful<br>-1 if file not found or in-use |

Explanation

The Set File Attributes function searches the directory for the file specified by the FCB drive, name, and type fields (bytes 0 through 11) and the current user number, and updates the file attributes in the directory from those in the FCB. (File attributes are stored in the high-order bit of FCB bytes 1-4 and 9-11.)

In addition, if FCB interface attribute f6 is set, this function updates the last record byte count of the file. The count is obtained from the current record field (byte 32) of the FCB, and stored in the specl field (byte 13) of each directory entry.

A program may set attributes on a file that it has open, in which case a close is performed implicitly before the attributes are set. However, a program is not permitted to set attributes on a file that another process has open.

---

**C-Function 31** **Get DPB Address**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 31 | |

Returned Value

| Reg | Description |
|---|---|
| ES:BX = DPB address | |

Explanation

The Get DPB Address function causes TurboDOS to construct a CP/M-style Disk Parameter Block (DPB) for the current drive, and to return its memory address in ES:BX.

NOTE: This function is supported only if the optional module CPMSUP is included during TurboDOS system generation.

---

**C-Function 32** **Get/Set User Number**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 32 | |
| DL = -1 to get user number | |
| 0-31 to set user number | |

Returned Value

| Reg | Description |
|---|---|
| AL = user number 0-31 (if get) | |

Explanation

The Get/Set User Number function can be used either to set or to return the current user number. If the value -1 (0xFF) is passed in register DL, this function returns the current user number in register AL. If some other value is passed in register DL and if the caller is a privileged log-on, this function sets the current user number to the specified value (modulo 32). A request to set the current user number from a non-privileged log-on is ignored.

**C-Function 33** **Read Random**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 33 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 1 | if reading unwritten data |
| 3 | if error changing extents |
| 4 | if reading unwritten extent |
| 6 | if random record number invalid |

Explanation

The Read Random function reads a 128-byte record from a file into memory at the current DMA address. The particular record to be read is specified by a 20-bit random record number obtained from FCB random record field (bytes 33 through 35). The given FCB must have been previously opened by an Open (15) or Make (22) C-function.

This function sets the FCB extent and current record fields to correspond with the record that was read. Unlike C-function 20 (Read Sequential), however, it does not increment the current record field after reading. Thus, if the Read Random function is followed by a Read Sequential or Write Sequential, the same record is re-accessed.

**C-Function 34** **Write Random**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 34 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 2 | if disk full or write-protected |
| 3 | if error changing extents |
| 5 | if no directory space |
| 6 | if random record number invalid |
| 8 | if writing locked record |

Explanation

The Write Random function writes a 128-byte record to a file from the current DMA address in memory. The particular record to be written is specified by a 20-bit random record number obtained from FCB random record field (bytes 33 through 35). The given FCB must have been previously opened by an Open (15) or Make (22) C-function.

This function sets the FCB extent and current record fields to correspond with the record that was written. Unlike C-function 21 (Write Sequential), however, it does not increment the current record field after writing. Thus, if the Write Random function is followed by a Read Sequential or Write Sequential, the same record is re-accessed.

**C-Function 35** **Compute File Size**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 35 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if file not found | |

Explanation

The Compute File Size function searches the directory for the file specified by the FCB drive, name, and type fields (bytes 0 through 11). If the file is found, this function sets the FCB random record field (bytes 33 through 35) to a value one greater than the record number of the last record in the file. Thus, a succeeding Write Random function (34) will append an additional record at the end of the file.

In TurboDOS, the Compute File Size function returns the correct result whether the file is open or closed.

---

**C-Function 36** **Set Random Record**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 36 | |
| DS:DX = FCB address | |

Explanation

The Set Random Record function returns the current file position of an open file in the random record field (bytes 33-35) of the FCB. (The file position is determined from the values of the FCB extent, spec2, and current record fields.) Since the Read Sequential (20) and Write Sequential (21) functions do not update the random record field of the FCB, this function is useful when switching from sequential to random access.

---

**C-Function 37** **Reset Drive**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 37 | |
| DX = reset vector | |

Explanation

The Reset Drive function write-enables the disk drives specified by the 16-bit reset vector passed in register DX. The reset vector contains a one-bit for each disk drive that is to be write-enabled, and a zero-bit for each drive that is not. The least significant bit corresponds to drive A, and the most significant bit to drive P.

NOTE: This function is supported only if the optional module CPMSUP is included during TurboDOS system generation.

---

**C-Function 40** **Write Random with Zero Fill**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 40 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 2 | if disk full or write-protected |
| 3 | if error changing extents |
| 5 | if no directory space |
| 6 | if random record number invalid |
| 8 | if writing locked record |

Explanation

The Write Random with Zero Fill function is implemented in TurboDOS as a synonym for Write Random (C-function 34).

**C-Function 42** **Lock Record**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 42 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 1 | if positioning to unwritten data |
| 3 | if error changing extents |
| 4 | if positioning to missing extent |
| 6 | if random record number invalid |
| 8 | if locked by another process |

Explanation

The Lock Record function attempts to obtain a lock on the record specified by a 20-bit random record number obtained from FCB random record field (bytes 33 through 35). The given FCB must have been previously opened in shared mode. If the file is not open in shared mode, this function performs no operation and returns a successful result.

The file is positioned to the specified record, unless compatibility flag bit 3 (logical) is set. If the specified record is already locked by another process, this function either suspends or returns an error (A=8) depending upon the setting of compatibility flag bit 6 (suspend).

If the FCB random record field is set to the 24-bit value 0xFFFFFF, then this function attempts to obtain an all-inclusive lock (on all records of the file at once). In this case, no positioning is performed.

**C-Function 43** **Unlock Record**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 43 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 1 | if positioning to unwritten data |
| 3 | if error changing extents |
| 4 | if positioning to missing extent |
| 6 | if random record number invalid |

Explanation

The Unlock Record function unlocks the record specified by a 20-bit random record number obtained from FCB random record field (bytes 33 through 35). Attempting to unlock a record which was not previously locked does not return an error. The given FCB must have been previously opened in shared mode. If the file is not open in shared mode, this function performs no operation and returns a successful result.

The file is positioned to the specified record, unless compatibility flag bit 3 (logical) is set.

If the FCB random record field is set to the 24-bit value 0xFFFFFF, then this function releases any all-inclusive lock on the file, but does not affect any individual record locks. In this case, no positioning is performed.

**C-Function 46** **Get Disk Free Space**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 46 | |
| DL = disk drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | |

Explanation

The Get Disk Free Space function determines the amount of free space on the specified disk drive. It returns a 24-bit binary value (the number of free 128-byte records) as a three byte quantity stored at the current DMA address, least significant byte first.

---

**C-Function 47** **Chain to Program**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 47 | |
| DL = 0 | to revert to orig. current disk |
| | -1 to retain present current disk |

Explanation

The Chain to Program function provides a means of chaining from one program to another. The calling program must place a valid TurboDOS command line, terminated by a null byte, in the Base Page record buffer starting at location 0x0080. This function terminates the calling program, and then executes the command line.

If DL = 0, the current disk reverts to what it was when the calling program was originally loaded into the TPA (the normal warm-start procedure). If DL = -1, however, the current disk at the time of call is retained.

**C-Function 50** **Direct BIOS Call**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 50 | |
| DS:DX = BIOS Parameter Block address | |

Returned Value

| Reg | Description |
|---|---|
| BX = BIOS return value | |

Explanation

The Direct BIOS Call function simulates a direct call to a CP/M BIOS routine. This function is called with the address of a BIOS Parameter Block in DS:DX. The BIOS Parameter Block is five bytes long, and has the following structure:

| Offset | Description |
|---|---|
| 0 | BIOS function number |
| 1-2 | CX-register entry value |
| 3-4 | DX-register entry value |

Under TurboDOS, such BIOS functions calls are emulated by converting them to an equivalent C-function call. Consequently, there is no performance advantage in using the Direct BIOS Call function, and its use is not encouraged.

The table on the next page describes the various simulated BIOS functions which may be invoked via the Direct BIOS Call function.

Explanation (Continued)

| BIOS Fcn# | Description | Equiv C-Fcn |
|---|---|---|
| 0 | Cold start | - |
| 1 | Warm start | 0 |
| 2 | Console status to AL | 11 |
| 3 | Raw console input to AL | 3 |
| 4 | Raw console output from CL | 4 |
| 5 | List output from CL | 5 |
| 6 | Raw console output from CL | 4 |
| 7 | Raw console input to AL | 3 |
| 8 | Set track to zero | - |
| 9 | Select disk drive from CL | 14 |
| 10 | Set track number from CX | - |
| 11 | Set sector number from CX | - |
| 12 | Set DMA offset from CX | 26 |
| 13 | Read disk sector ($.DSK) | 33 |
| 14 | Write disk sector ($.DSK) | 34 |
| 15 | List status to AL (always -1) | - |
| 16 | Sector translate CX into BX | - |
| 17 | Set DMA base from CX | 51 |
| 18 | MEMTBL offset to BX | - |
| 19 | Get IOBYTE to AL | 7 |
| 20 | Set IOBYTE from CL | 8 |

**C-Function 51** **Set DMA Base**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 51 | |
| DX = DMA base (paragraph address) | |

Explanation

The Set DMA Base function causes the paragraph address specified in register DX to be used in conjunction with the current DMA offset as the record buffer address for subsequent read and write operations. (See C-function 26, Set DMA Offset.)

Whenever a program is loaded into the TPA, the DMA base is initialized to the initial data segment base.

---

**C-Function 52** **Return DMA Address**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 52 | |

Returned Value

| Reg | Description |
|---|---|
| BX = current DMA offset | |
| ES = current DMA base | |

Explanation

The Return DMA Address function returns the current DMA base (paragraph address) in ES and the current DMA offset (byte address) in BX.

---

**C-Function 53** **Allocate Maximum Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 53 | |
| DS:DX = MCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if no memory was available | |

Explanation

The location of a Memory Control Block (MCB) is passed in DS:DX. The MCB is five bytes long and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | MCB-Base (paragraph address) |
| 2-3 | MCB-Length (in paragraphs) |
| 4 | MCB-Ext (byte value) |

The Allocate Maximum Memory function allocates the largest available memory region of size less than or equal to the number of paragraphs specified by MCB-Length. If successful, the base address and length of the allocated region are returned in MCB-Base and MCB-Length, and MCB-Ext is set to 1.

---

**C-Function 54** **Allocate Absolute Maximum Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 54 | |
| DS:DX = MCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| -1 | if no memory was available |

Explanation

The location of a Memory Control Block (MCB) is passed in DS:DX. The MCB is five bytes long and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | MCB-Base (paragraph address) |
| 2-3 | MCB-Length (in paragraphs) |
| 4 | MCB-Ext (byte value) |

The Allocate Absolute Maximum Memory function is not supported by TurboDOS, and always returns AL = -1.

**C-Function 55** **Allocate Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 55 | |
| DS:DX = MCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| -1 | if memory was not available |

Explanation

The location of a Memory Control Block (MCB) is passed in DS:DX. The MCB is five bytes long and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | MCB-Base (paragraph address) |
| 2-3 | MCB-Length (in paragraphs) |
| 4 | MCB-Ext (byte value) |

The Allocate Memory function attempts to allocate a memory region of the size specified by MCB-Length. If successful, the base address of the allocated region is returned in MCB-Base.

---

**C-Function 56** **Allocate Absolute Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 56 | |
| DS:DX = MCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful<br>-1 if memory was not available | |

Explanation

The location of a Memory Control Block (MCB) is passed in DS:DX. The MCB is five bytes long and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | MCB-Base (paragraph address) |
| 2-3 | MCB-Length (in paragraphs) |
| 4 | MCB-Ext (byte value) |

The Allocate Absolute Memory function is not supported by TurboDOS, and always returns AL = -1.

**C-Function 57** **Free Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 57 | |
| DS:DX = MCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if invalid request | |

Explanation

The location of a Memory Control Block (MCB) is passed in DS:DX. The MCB is five bytes long and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | MCB-Base (paragraph address) |
| 2-3 | MCB-Length (in paragraphs) |
| 4 | MCB-Ext (byte value) |

The Free Memory function is used to deallocate memory regions previously allocated by the calling program. If MCB-Ext is passed as -1, then all memory allocated by the calling program and its descendants is deallocated. If MCB-Ext is passed as 0, then just the region defined by MCB-Base and MCB-Length is deallocated. In the latter case, either the starting address or the ending address (or both) of the specified region must be equal to that of a region previouly allocated by the calling program.

---

**C-Function 58** **Free All Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 58 | |

Explanation

The Free All Memory function deallocates all previously allocated memory regions, regardless of who allocated them.

TurboDOS automatically performs this function at each program termination (warm-start), so it is almost never necessary for a program to call this function explicitly.

**C-Function 59** **Program Load**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 59 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| BX = Base Page paragraph address or 0xFFFF if unsuccessful | |

Explanation

The Program Load function loads the .CMD file specified by the FCB into the TPA. Memory regions are automatically allocated for each segment group as specified by the .CMD header record, and code or data is loaded as required into the allocated regions from the body of the .CMD file.

Note that this function does not affect the current DMA base or offset addresses.

**C-Function 104** **Set Date and Time**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 104 | |
| DS:DX = date/time packet address | |

Explanation

The Set Date and Time function sets the system date and time. The address of a four-byte date/time packet is passed in DS:DX. The date/time packet has the following structure:

| Offset | Description |
|---|---|
| 0-1 | Date, represented as a 16-bit Julian date with zero corresponding to 31 December 1977. |
| 2 | Hours, represented as two binary coded decimal (BCD) digits |
| 3 | Minutes, represented as two binary coded decimal (BCD) digits |

Seconds are set to zero.

**C-Function 105** **Get Date and Time**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 105 | |
| DS:DX = date/time packet address | |

Returned Value

| Reg | Description |
|---|---|
| AL = seconds (two BCD digits) | |

Explanation

The Get Date and Time function returns the system date and time in a four-byte date/time packet whose address is passed in DS:DX. The date/time packet has the following structure:

| Offset | Description |
|---|---|
| 0-1 | Date, represented as a 16-bit Julian date with zero corresponding to 31 December 1977. |
| 2 | Hours, represented as two binary coded decimal (BCD) digits |
| 3 | Minutes, represented as two binary coded decimal (BCD) digits |

Seconds are returned in register AL, represented as two binary coded decimal (BCD) digits.

---

**C-Function 107** **Return Serial Number**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 107 | |
| DS:DX = address of 6-byte S/N field | |

Explanation

The Return Serial Number function returns the CP/M serial number in the 6-byte field whose address is passed in DS:DX. Under TurboDOS, this function always returns six zero bytes.

NOTE: This function is supported only if the optional module CPMSUP is included during TurboDOS system generation.

---

**C-Function 108** **Get/Set Program Return Code**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 108 | |
| DX = 0xFFFF (if get) | |
| | program return code (if set) |

Returned Value

| Reg | Description |
|---|---|
| BX = program return code (if get) | |

Explanation

The Get/Set Program Return Code function provides a means for one program to pass a 16-bit value to another program. For example, this function can be used to advantage in connection with C-function 47 (Chain to Program).

If register DX is set to 0xFFFF, then this function interrogates the program return code and returns it in register BX. Otherwise, this function sets the program return code to the value passed in register DX.

---

**C-Function 110** **Get/Set Program Output Delimiter**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 110 | |
| DX = 0xFFFF (if get), or | |
| DL = output delimiter (if set) | |

Returned Value

| Reg | Description |
|---|---|
| AL = output delimiter (if get) | |

Explanation

The Get/Set Output Delimiter function can be used to set or interrogate the output delimiter used by C-function 9 (Print String). Whenever a program is loaded into the TPA, the output delimiter is initialized to the dollar sign $ character.

If register DX is set to 0xFFFF, then this function interrogates the current output delimiter and returns it in register AL. Otherwise, this function sets the output delimiter to the value passed in register DL.

---

**C-Function 111** **Print Block**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 111 | |
| DS:DX = CCB address | |

Explanation

The Print Block function displays a string of characters on the console screen. The string may be of any length, and is defined by a Character Control Block (CCB) whose address is passed in DS:DX. The CCB is four bytes long, and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | starting DS-offset of string |
| 2-3 | byte-length of string |

Horizontal tabs are expanded into multiple spaces, based upon tab stops at every eighth column.

---

**C-Function 112** **List Block**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 112 | |
| DS:DX = CCB address | |

Explanation

The List Block function sends a string of characters to be printed according to the current print routing. The string may be of any length, and is defined by a Character Control Block (CCB) whose address is passed in DS:DX. The CCB is four bytes long, and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | starting DS-offset of string |
| 2-3 | byte-length of string |

Horizontal tabs are not expanded.

**C-Function 152** **Parse Filename**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 152 | |
| DS:DX = PFCB address | |

Returned Value

| Reg | Description |
|---|---|
| BX = 0 if successful and end of line<br>0xFFFF if error while parsing<br>delimiter offset otherwise | |

Explanation

The Parse Filename function parses an ASCII file specification of the form:

{d:}filename{.typ}

into FCB format. The FCB drive, name, and type fields (bytes 0 through 11) are initialized according to the parsed file specification. Bytes 12 through 15 of the FCB are zeroed.

This function is called with the address of a Parse Filename Control Block (PFCB) in DS:DX. The PFCB is four bytes long, and has the following structure:

| Offset | Description |
|---|---|
| 0-1 | DS-offset of ASCII input string |
| 2-3 | DS-offset of destination FCB |

This function parses the first file specification it finds in the input string. Leading spaces are ignored. Parsing stops upon encountering a space, comma, semicolon, equal-sign, or any ASCII control character.

(Intentionally left blank.)

## T-FUNCTIONS

This section describes the 43 TurboDOS-unique functions ("T-functions") which supplement the C-functions described in the previous section. The T-functions are presented in numerical order, with calling parameters, return values, and a detailed explanation for each.

To invoke a T-function, a program executes the interrupt instruction INT 0xE1 with a function number in register CL. Arguments are passed and values returned in registers, as described below for each T-function.

If a T-function call is made with register CL set to an unsupported function number, TurboDOS returns immediately with register AX set to zero.

T-function calls generally destroy registers AX-BX-CX-DX-SI-DI-BP-ES, but preserve SP-IP and CS-DS-SS.

---

**T-Function 0** **Reset Operating System**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 0 | |

Explanation

The Reset Operating System function unlocks all locked records, closes all open files, unlocks all locked drives, and terminates any network sessions involving the calling process.

TurboDOS automatically performs this function at each program termination (warm-start), so it is almost never necessary for a program to call this function explicitly.

**T-Function 1** **Create Process**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 1 | |
| DX = entrypoint offset | |
| BX = workspace offset | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if insufficient memory | |

Explanation

The Create Process function creates a new process which starts execution at the entry-point offset passed in register DX. The new process is assigned a TurboDOS work area whose offset appears to the new process in register SI, and a 64-word stack area whose offset appears in register SP. If the process requires a re-entrant work area (usually allocated dynamically using T-function 3), its offset should be passed in register BX and will appear to the new process in register DI.

If this function is called with register DX set to zero, it causes the calling process to terminate.

NOTE: This function is intended to be invoked only by resident processes within TurboDOS. It deals with 16-bit offset values that are relative to the operating system base. Consequently, it should never be invoked from a transient program.

---

**T-Function 2** **Delay Process**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 2 | |
| DX = tick count | |

Explanation

The Delay Process function causes the calling process to be suspended for the period of time specified by the tick count passed in register DX. A system "tick" is an implementation-dependent time interval, usually 1/50 or 1/60 of a second. The actual delay may vary from the requested tick count by plus or minus one tick.

If the specified tick count is zero, then the calling program is suspended only long enough to allow any other ready processes to run (a so-called "courtesy" dispatch).

---

**T-Function 3** **Allocate Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 3 | |
| DX = byte-length of requested segment | |

Returned Values

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if insufficient memory | |
| BX = segment offset (if successful) | |

Explanation

The Allocate Memory function allocates a contiguous memory segment of the byte-length requested in register DX. If successful, the starting offset of the allocated segment is returned in register BX.

NOTE: This function is intended to be invoked only by resident processes within TurboDOS. It deals with 16-bit offset values that are relative to the operating system base. Consequently, it should never be invoked from a transient program. If a memory segment is allocated by a process and not deallocated before the process terminates, then the space is lost permanently.

**T-Function 4** **Deallocate Memory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 4 | |
| DX = segment offset | |

Explanation

The Deallocate Memory function returns a previously-allocated memory segment to the pool of available memory space.

NOTE: This function is intended to be invoked only by resident processes within TurboDOS. It deals with 16-bit offset values that are relative to the operating system base. Consequently, it should never be invoked from a transient program. The offset passed in DX must be a segment starting offset returned by a prior call to C-function 3 (Allocate Memory), otherwise a system crash may occur.

**T-Function 5** **Send Interprocess Message**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 5 | |
| DX = message node offset | |
| BX = message offset | |

Explanation

The Send Interprocess Message function provides a means to send messages from one process to another. Register DX specifies the offset of a 10-byte message node which must be initialized as follows:

```
MSGNOD: WORD  0         ;semaphore count
        WORD  MSGNOD+2  ;semaphore head
        WORD  MSGNOD+2  ;     "       "
        WORD  MSGNOD+4  ;msg chain head
        WORD  MSGNOD+4  ; "    "    "
```

Register BX specifies the offset of the message to be sent, which must be prefixed by a 4-byte linkage as follows:

```
MESSAG: WORD  0         ;message linkage
        WORD  0         ;     "       "
        BYTE  ...       ;message text
        BYTE  ...       ;(any length)
```

NOTE: This function is intended to be invoked only by resident processes within TurboDOS. It deals with 16-bit offset values that are relative to the operating system base. Consequently, it should never be invoked from a transient program.

---

**T-Function 6** **Receive Interprocess Message**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 6 | |
| DX = message node offset | |

Returned Value

| Reg | Description |
|---|---|
| BX = message offset | |

Explanation

The Receive Interprocess Message function provides a means to receive messages sent by another process using C-function 5 (Send Interprocess Message). Register DX specifies the offset of a 10-byte message node which must be initialized as follows:

```
MSGNOD: WORD  0         ;semaphore count
        WORD  MSGNOD+2  ;semaphore head
        WORD  MSGNOD+2  ;     "      "
        WORD  MSGNOD+4  ;msg chain head
        WORD  MSGNOD+4  ; "    "    "
```

If no message is available from the specified message node, the calling process is suspended until a message arrives. This function returns in BX the offset of the received message prefixed by a 4-byte linkage.

NOTE: This function is intended to be invoked only by resident processes within TurboDOS. It deals with 16-bit offset values that are relative to the operating system base. Consequently, it should never be invoked from a transient program.

**T-Function 7** **Set Error Address**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 7 | |
| DX = | error intercept routine offset, or 0 to restore default error handling |
| BX = | error intercept routine base |

Explanation

The Set Error Address function enables a program to establish its own error intercept routine to intercept and process unrecoverable disk errors. The address of the intercept routine is passed in BX (base) and DX (offset). Normal TurboDOS error diagnosis is supressed.

The error intercept routine must not call any TurboDOS functions, and must return via a RETF instruction with register AL set to the desired error recovery alternative:

| AL-reg | Recovery Action |
|---|---|
| 0 | retry operation |
| +1 | ignore error |
| -1 | abort program |

If the Set Error Address function is called with DX set to zero, normal TurboDOS error diagnosis is restored. This also happens automatically when the program terminates.

**T-Function 8** **Set Abort Address**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 8 | |
| DX = | abort intercept routine offset, or 0 to restore default abort handling |
| BX = | abort intercept routine base |

Explanation

The Set Abort Address function enables a program to establish its own abort intercept routine to intercept and process user-requested aborts (in response to attention-requests or disk errors). The address of the intercept routine is passed in BX (base) and DX (offset).

The abort intercept routine may exit via a RETF instruction to resume execution of the program at the point of interruption. Alternatively, it may proceed with any desired wrap-up processing and then terminate the program (via C-function 0).

If the Set Abort Address function is called with DX set to zero, normal TurboDOS abort handling restored. This also happens automatically when the program terminates.

---

**T-Function 9** **Set Date and Time**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 9 | |
| BX = Julian date (0 is 31 December 1947) | |
| DH = hours (0-23, binary integer) | |
| DL = minutes (0-59, binary integer) | |
| CH = seconds (0-59, binary integer) | |

Explanation

The Set Date and Time function sets the system date and time. The Julian date passed in register BX is the number of days since the base date of 31 December 1947. Dates prior to the base date are represented by negative values.

The system date and time may also be set by means of C-function 104 (Set Date and Time), but the format of arguments is considerably different.

**T-Function 10** **Get Date and Time**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 10 | |

Returned Values

| Reg | Description |
|---|---|
| BX = Julian date (0 is 31 December 1947) | |
| DH = hours (0-23, binary integer) | |
| DL = minutes (0-59, binary integer) | |
| CH = seconds (0-59, binary integer) | |
| CL = system tick count | |

Explanation

The Get Date and Time function returns the system date and time. The Julian date returned in register BX is the number of days since the base date of 31 December 1947. Dates prior to the base date are represented by negative values.

The system tick count returned in register CL is incremented every system tick. It counts from zero to 255, then wraps around to zero. A system tick is an implementation-dependent time interval, usually 1/50 or 1/60 of a second.

The system date and time may also be interrogated by means of C-function 105 (Get Date and Time), but the format of returned values is considerably different.

**T-Function 11** **Rebuild Disk Map**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 11 | |
| DL = disk drive: | |
| 0 for drive A | |
| 1 for drive B | |
| : | |
| 15 for drive P | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if disk write-protected | |
| or has files open | |

Explanation

The Rebuild Disk Map function regenerates the allocation map on the disk drive specified in register DL. The principal purpose of this function is to support the FIXMAP command.

**T-Function 12** **Return Serial Number**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 12 | |

Returned Values

| Reg | Description |
|---|---|
| BX = TurboDOS origin number | |
| DX = TurboDOS unit number | |
| CH = 0 if non-privileged log-on | |
| 0x80 if privileged log-on | |
| CL = 0x14 (TurboDOS version 1.3) | |

Explanation

The Return Serial Number function returns the origin and unit numbers with which this particular copy of TurboDOS was serialized, and may be used in application programs to help prevent unauthorized use.

This function also returns the TurboDOS version number, and a flag which indicates whether or not the current log-on is privileged.

**T-Function 13** **Set Compatibility Flags**

Entry Arguments

| Reg | Description |
|---|---|
| | |
| CL = 13 | |
| DL = compatibility flags: | |
| | bit 7 = permissive flag |
| | bit 6 = suspend flag |
| | bit 5 = global-write flag |
| | bit 4 = mixed-mode flag |
| | bit 3 = logical flag |
| | (bits 2-0 not defined) |

Explanation

The Set Compatibility Flags function enables a program to modify the rules by which file sharing is done. The meaning of each compatibility flag is described in section 2.

When the program terminates, the compatibility flags revert automatically to the default values assigned to the public symbol COMPAT at system generation.

---

**T-Function 14** **Log-On/Log-Off**

Entry Arguments

| Reg | Description |
|---|---|
| CL = | 14 |
| DX = | 0xFFFF (if log-off) |
| DL = | user number 0-31 (if log-on)<br>with bit 7 set for privileged |
| DH = | current disk drive (if log-on):<br>-1 for no change<br>0 for drive A<br>1 for drive B<br>:<br>15 for drive P |

Returned Value

| Reg | Description |
|---|---|
| AL = | 0 if successful<br>-1 if request invalid |

Explanation

The Log-On/Log-Off function is provided to support log-on security via the LOGON and LOGOFF commands. To log-on, this function is called with the desired user number in register DL (with bit 7 set if a privileged log-on is desired), and with the desired current drive in register DH (or -1 for no change in current drive). To log-off, the function is called with DX set to 0xFFFF.

After a log-off, another log-on request is not honored until a warm-start or C-function 0 (System Reset) has occurred.

NOTE: When this function is called from a resident system process, the argument in DH is ignored.

**T-Function 15** **Load File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 15 | |
| DS:DX = FCB address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| 1 | if not enough memory to load file |
| -1 | if file not found |

Explanation

The Load File function loads the file specified by the FCB drive, name, and type fields (bytes 0 through 11) into memory starting at the current DMA address. The file need not have been opened. If the top of the TPA is reached before the end-of-file is encountered, the loading stops and an error is returned.

Note that this function does not allocate TPA space or interpret a .CMD header. Use C-function 59 (Program Load) to load programs and overlays stored in .CMD format.

**T-Function 16** **Activate Do-File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 16 | |
| DS:DX = FCB address (to activate) | |
| DX = 0 (to cancel) | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if file not found | |

Explanation

The Activate Do-File function causes the file specified by the FCB drive, name, and type fields (bytes 0 through 11) to be activated as a do-file. The file need not have been opened. Any currently-active do-file and/or command line is stacked (to be reactivated when the new do-file has been processed to completion). The principal purpose of this function is to support the DO command.

This function may also be called with DX set to zero to cancel all active and stacked do-files.

**T-Function 17** **Disable/Enable Autoload**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 17 | |
| DL = 0 | to disable autoload |
| -1 | to enable autoload |

Explanation

The Disable/Enable Autoload function may be used to disable the warm-start autoload feature of TurboDOS, or to re-enable the feature after it has been disabled.

TurboDOS automatically disables the warm-start autoload feature whenever it fails to find the file WARMSTRT.AUT on the current disk during a warm-start. Creating such a file on disk (or changing the current disk to one that contains such a file) will not result in autoloading unless the autoload feature is explicitly re-enabled by means of this function.

---

**T-Function 18** **Send Command Line**

Entry Arguments

| Reg | Description |
|---|---|
| CL | = 18 |
| DS:DX | = buffer address (to send) |
| DX | = 0 (to cancel) |

Explanation

The Send Command Line function allows a program to specify the next command line to be processed by TurboDOS after the program terminates. The buffer address is passed in DS:DX. The first byte of the buffer must contain the command line byte-length, and the command line text must occupy the second and succeeding bytes of the buffer. Any currently-active command line is stacked, and the new command line is activated.

This function may also be called with DX set to zero to cancel all active and stacked command lines.

**T-Function 19** **Return Disk Allocation Information**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 19 | |
| DL = disk drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |

Returned Values

| Reg | Description |
|---|---|
| AL = block size: | |
| | 3 for 1K blocks |
| | 4 for 2K blocks |
| | : |
| | 7 for 16K blocks |
| | plus: bit 7 set if fixed disk |
| | bit 6 set if EXM=0 forced |
| CL = number of blocks in the directory | |
| DX = number of blocks presently unused | |
| BX = total number of blocks on the disk | |

Explanation

The Return Disk Allocation Information function returns various parameters concerning the logical organization of the specified disk drive.

**T-Function 20** **Return Physical Disk Information**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 20 | |
| DL = disk drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |

Returned Values

| Reg | Description |
|---|---|
| AL = physical sector size: | |
| | 0 for 128-byte sectors |
| | 1 for 256-byte sectors |
| | 2 for 512-byte sectors |
| | 3 for 1K sectors |
| | : |
| | 7 for 16K sectors |
| CX = number of reserved (boot) tracks | |
| DX = total number of tracks on the disk | |
| BX = number of sectors per track | |

Explanation

The Return Physical Disk Information function returns various parameters concerning the format and physical organization of the specified disk drive.

## T-Function 21 Get/Set Drive Status

Entry Arguments

| Reg | Description |
|---|---|
| CL = | 21 |
| DL = | disk drive:<br>0 for drive A<br>1 for drive B<br>:<br>15 for drive P |
| DH = | 0 to set the drive read/write<br>1 to set the drive read-only<br>-1 to return the drive status |

Returned Values

| Reg | Description |
|---|---|
| AL = | 0 if successful<br>-1 if attempt to set drive status while files are open |
| BL = | 0 if drive is not ready<br>-1 if drive is ready |
| BH = | 0 if drive is read/write<br>-1 if drive is read-only |

Explanation

The Get/Set Drive Status function may be used to interrogate the ready and write-protect status of the drive specified by register DL. This function may also be used to change the write-protect status of the drive. The code passed in register DH controls which of these operations is performed, as indicated above.

**T-Function 22** **Physical Disk Access**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 22 | |
| DS:DX = PDR packet address | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if read/write successful, or drive not ready<br>-1 if read/write unsuccessful, or drive is ready | |

Explanation

The Physical Disk Access function provides direct access to the physical disk drivers. The principal purpose of this function is to support the BOOT, BACKUP, FORMAT, and VERIFY commands. It is honored for privileged log-ons only, and may be used only for disk drives local to the calling processor.

DS:DX contains the address of a 16-byte physical disk request (PDR) packet with the following structure:

| Offset | Description |
|---|---|
| 0 | disk operation code (0-4) |
| 1 | disk drive (0-15) |
| 2-3 | physical track number (base 0) |
| 4-5 | physical sector number (base 0) |
| 6-7 | number of sectors to read/write |
| 8-9 | number of bytes to read/write |
| 10-11 | DMA offset for read/write |
| 12-13 | DMA base (para) for read/write |
| 14-15 | disk specification table address |

The physical operation performed depends upon the disk operation code in the PDR packet.

Explanation (Continued)

If the PDR opcode is 0, the specified number of physical sectors (or bytes) are read from the specified drive, track, and sector into the specified DMA address.

If the PDR opcode is 1, the specified number of physical sectors (or bytes) are written to the specified drive, track, and sector from the specified DMA address.

If the PDR opcode is 2, the type of the specified disk is determined, and an 11-byte disk specification table (DST) is returned at the specified DMA address, structured as follows:

| Offset | Description |
|---|---|
| 0 | block size (3=1K,4=2K,...,7=16K) |
| 1-2 | total number of blocks on disk |
| 3 | number of directory blocks |
| 4 | sector size (0=128,...,7=16K) |
| 5-6 | number of sectors per track |
| 7-8 | number of tracks on the disk |
| 9-10 | number of reserved (boot) tracks |

If the PDR opcode is 3, the ready status of the specified drive is returned in register AL (0 if not ready, -1 if ready).

If the PDR opcode is 4, the specified track of the specified drive is formatted, using hardware-dependent formatting information provided at the specified DMA address.

NOTE: Opcodes 0 (read) and 1 (write) require that the PDR packet contain the address of a valid DST for the specified disk. Therefore, opcode 2 (return DST) should be invoked first to obtain the DST.

---

**T-Function 23** **Set Buffer Parameters**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 23 | |
| DH = number of buffers (minimum 2) | |
| DL = buffer size: | |
| | 0 for 128-byte buffers |
| | 1 for 256-byte buffers |
| | 2 for 512-byte buffers |
| | 3 for 1K buffers |
| | : |
| | 7 for 16K buffers |

Explanation

The Set Buffer Parameters function enables the number and size of disk buffers to be changed. The principal purpose of this function is to support the BUFFERS command.

The specified number of buffers must be at least 2. If the specified number of buffers cannot be allocated due to insufficient memory, then TurboDOS allocates as many as it can. The specified buffer size must be as least as large as the largest physical disk sector size being used.

If this function is called from a slave processor without local disk storage, then the function is passed over the network to be processed in the master.

**T-Function 24** **Get Buffer Parameters**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 24 | |

Returned Values

| Reg | Description |
|---|---|
| BH = number of buffers | |
| BL = buffer size: | |
| | 0 for 128-byte buffers |
| | 1 for 256-byte buffers |
| | 2 for 512-byte buffers |
| | 3 for 1K buffers |
| | : |
| | 7 for 16K buffers |

Explanation

The Get Buffer Parameters function enables the number and size of disk buffers to be interrogated. The principal purpose of this function is to support the BUFFERS command.

If this function is called from a slave processor without local disk storage, then the function is passed over the network to be processed in the master.

---

**T-Function 25** **Lock/Unlock Drive**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 25 | |
| DL = disk drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |
| DH = 0 to unlock drive | |
| | -1 to lock drive |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| | -1 if drive in-use or already locked by another process |

Explanation

The Lock/Unlock Drive function enables a program to secure a lock on a specified disk drive. This function is used by many TurboDOS commands such as BACKUP, CHANGE, FIXDIR, FIXMAP, FORMAT, and VERIFY to ensure that they cannot compromise the processing of other users.

**T-Function 26** **Flush/Free Buffers**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 26 | |
| DL = disk drive: | |
| 0 for drive A | |
| 1 for drive B | |
| : | |
| 15 for drive P | |
| DH = subfunction flags: | |
| bit 7 set | to free buffers unconditionally |
| bit 6 set | to free buffers after disk error abort |
| bit 5 set | to continue after disk error abort |
| bit 4 set | to return after disk error abort |

Explanation

The Flush/Free Buffers function causes all written-to disk buffers for the specified disk drive to be written out (flushed) to the disk. This function may cause disk buffers for the specified drive to be freed, conditionally or unconditionally, according to the subfunction flags passed in register DH.

It is suggested that this function be used prior to media changes and physical disk access (T-function 22).

---

**T-Function 27** **Get/Set Print Mode**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 27 | |
| DL = print mode: | |
| | 0 to print direct |
| | 1 to print spooled |
| | 2 to print to the console |
| | -1 to leave print mode unchanged |
| DH = printer assignment (if mode = 0) | |
| queue assignment (if mode = 1) | |
| -1 to leave assignment unchanged | |
| CH = spool drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |
| | -1 to leave spool drive unchanged |

Returned Values

| Reg | Description |
|---|---|
| If B = D = E = -1 on entry, returns with: | |
| AL = current spool drive | |
| BH = current printer or queue assignment | |
| BL = current print mode | |

Explanation

The Get/Set Print Mode function is used to set or interrogate print routing, and is provided to support the PRINT command.

Printer and queue assignments are coded thus: 1 for A, 2 for B, ..., 16 for P. Assignment to queue zero causes print files to be left unqueued. Assignment to printer zero causes print output to be discarded. Setting the assignment, mode, or spool drive implies an immediate end-of-print-job condition. If registers CH, DH, and DL are all set to -1, this function simply interrogates and returns the current assignment, mode, and drive.

**T-Function 28** **Signal End-of-Print**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 28 | |

Explanation

The Signal End-of-Print function causes an end-of-print condition. If spooling is in effect, the current print file is closed and (if appropriate) enqueued for background printing.

An end-of-print condition may also occur as the result of a warm-start, attention request, or end-of-print character.

**T-Function 29** **Get/Set De-Spool Mode**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 29 | |
| CH = printer: | |
| | 0 for printer A |
| | 1 for printer B |
| | : |
| | 15 for printer P |
| DL = de-spool mode: | |
| | 0 to process print job |
| | 1 to suspend print job |
| | 2 to begin print job over |
| | 3 to terminate print job |
| | -1 to leave mode unchanged |
| DH = de-spool queue assignment: | |
| | 0 to set printer off-line |
| | 1 for queue A |
| | 2 for queue B |
| | : |
| | 16 for queue P |
| | -1 to leave queue unchanged |

Returned Values

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if invalid request | |
| If D = E = -1 on entry, returns with: | |
| BH = current queue assignment (0-16) | |
| BL = current de-spool mode (0 or 1) | |

Explanation

The Get/Set De-Spool Mode function is used to control background printing, and is provided to support the PRINTER command. If registers DH and DL are both set to -1, this function simply interrogates and returns the current queue assignment and de-spool mode for the specified printer.

**T-Function 30** **Queue a Print File**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 30 | |
| DS:DX = FCB address | |
| BH = print queue: | |
| 0 for queue A | |
| 1 for queue B | |
| : | |
| 15 for queue P | |
| BL = user number (0-31), plus | |
| bit 7 set to delete after printing | |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 if successful | |
| -1 if invalid request | |

Explanation

The Queue a Print File function enqueues a text file on a specified print queue for background printing. The file to be enqueued is identified by the FCB drive, name and type fields (bytes 0 through 11), together with the user number passed in register BL.

The drive specified by the FCB must be accessible by the processor in which the specified queue resides, otherwise the request is invalid. To check this, the function may be called with register BL set to -1, in which case the FCB drive and requested queue are checked for validity but no file is queued.

---

**T-Function 31** **Flush List Buffer**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 31 | |

Explanation

The Flush List Buffer function is used by TurboDOS during direct printing over the network to force any remaining buffered characters to be printed. There should be no need for an application program to call this function.

---

**T-Function 32** **Network List Out**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 32 | |
| DL = output character | |

Explanation

The Network List Out function is used by TurboDOS during direct printing over the network. There should be no need for an application program to call this function.

---

**T-Function 33** **Remote Console I/O**

Entry Arguments

| Reg | Description |
|---|---|
| CL = | 33 |
| DL = | console input character, or<br>0 if no console input available |
| DH = | 0 to detach remote console<br>-1 to attach remote console |

Returned Value

| Reg | Description |
|---|---|
| AL = | 0 if CONREM not present<br>1 if successful<br>-1 if executing in master |

Explanation

The Remote Console I/O function works in conjunction with the CONREM console driver to support the MASTER command. It passes one byte of console input in register DL (if available), and returns a count byte and up to 127 bytes of console output at the current DMA address. There should be no need for an application program to call this function.

**T-Function 34** **Get Comm Channel Status**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 34 | |
| DH = | channel number, plus bit 7 set if remote channel |

Returned Value

| Reg | Description |
|---|---|
| AL = | 0 if input character not available<br>-1 if input character is available |

Explanation

The Get Comm Channel Status function checks to see whether or not an input character is available on the specified comm channel. If a character is available, it returns A = -1. Otherwise, it returns A = 0.

---

**T-Function 35** **Comm Channel Input**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 35 | |
| DH = channel number, plus bit 7 set if remote channel | |

Returned Value

| Reg | Description |
|---|---|
| AL = input character | |

Explanation

The Comm Channel Input function obtains the next input character from the specified comm channel, and returns in in register AL. If no character is available, the calling program is suspended until a character is received.

**T-Function 36** **Comm Channel Output**

Entry Arguments

| Reg | Description |
|---|---|
| CL = | 36 |
| DH = | channel number, plus bit 7 set if remote channel |
| DL = | output character |

Explanation

The Comm Channel Output function outputs the character passed in register DL on the specified comm channel.

**T-Function 37** **Set Comm Baud Rate**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 37 | |
| DH = channel number, plus | |
| bit 7 set if remote channel | |
| DL = baud rate code (bits 3-0): | |
| 0 for 50 baud | 8 for 1800 baud |
| 1 for 75 baud | 9 for 2000 baud |
| 2 for 110 baud | 10 for 2400 baud |
| 3 for 134.5 baud | 11 for 3600 baud |
| 4 for 150 baud | 12 for 4800 baud |
| 5 for 300 baud | 13 for 7200 baud |
| 6 for 600 baud | 14 for 9600 baud |
| 7 for 1200 baud | 15 for 19200 baud |
| plus bit 7 set for att'n detection | |
| bit 6 set for CTS handshaking | |
| bit 5 set for input disabled | |

Explanation

The Set Comm Baud Rate function sets the baud rate and options passed in register DL on the specified comm channel.

**T-Function 38** **Get Comm Baud Rate**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 38 | |
| DH = channel number, plus bit 7 set if remote channel | |

Returned Value

| Reg | Description |
|---|---|
| AL = baud rate code (bits 3-0): | |
| 0 for 50 baud | 8 for 1800 baud |
| 1 for 75 baud | 9 for 2000 baud |
| 2 for 110 baud | 10 for 2400 baud |
| 3 for 134.5 baud | 11 for 3600 baud |
| 4 for 150 baud | 12 for 4800 baud |
| 5 for 300 baud | 13 for 7200 baud |
| 6 for 600 baud | 14 for 9600 baud |
| 7 for 1200 baud | 15 for 19200 baud |
| plus bit 7 set for att'n detection | |
| bit 6 set for CTS handshaking | |
| bit 5 set for input disabled | |

Explanation

The Set Comm Baud Rate function interrogates the baud rate and options for the specified comm channel, and returns this information in register AL.

---

**T-Function 39** **Set Modem Controls**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 39 | |
| DH = channel number, plus | |
| | bit 7 set if remote channel |
| DL = modem control vector: | |
| | bit 7 set for request-to-send |
| | bit 6 set for data-terminal-ready |
| | bits 5-0 unassigned |

Explanation

The Set Modem Controls function sets the modem control signals in accordance with the vector passed in register DL on the specified comm channel.

---

**T-Function 40** **Get Modem Status**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 40 | |
| DH = channel number, plus bit 7 set if remote channel | |

Returned Value

| Reg | Description |
|---|---|
| AL = modem status vector: bit 7 set for clear-to-send | |
| bit 6 set for data-set-ready | |
| bit 5 set for data-carrier-detect | |
| bit 4 set for ring-indicator | |
| bits 3-0 unassigned | |

Explanation

The Set Modem Status function interrogates the modem status signals for the specified comm channel, and returns this information as a vector in register AL.

**T-Function 41** **User-Defined Function**

Entry Arguments

| Reg | Description |
|---|---|
| CL = | 41 |
| CH = | network routing:<br>0 if always processed locally<br>ld hex if routed per drive d<br>2p hex if routed per printer p<br>3q hex if routed per queue q<br>-1 if routed to default net addr |
| DX = | user-defined argument passed |
| BX = | user-defined argument passed |

Returned Values

| Reg | Description |
|---|---|
| AX = | user-defined value returned |
| CX = | user-defined value returned |
| DX = | user-defined value returned |
| BX = | user-defined value returned |

Explanation

The User-Defined Function provides a means for adding user-defined extensions to the operating system taking full advantage of the TurboDOS networking facilities. On entry, register CH defines how the request is to be routed over the network. Registers DX and BX plus the 128-byte record at the current DMA address are all passed (over the network if necessary) to a user-defined module with the public entrypoint symbol USRFCN. Upon entry to the USRFCN routine, register CX contains the DS-offset of the 128-byte record that was passed. The USRFCN routine may return information to the caller in any of the registers AL-BX-CX-DX and in the 128-byte record.

**T-Function 42** **Reorganize Disk Directory**

Entry Arguments

| Reg | Description |
|---|---|
| CL = 42 | |
| DL = disk drive: | |
| | 0 for drive A |
| | 1 for drive B |
| | : |
| | 15 for drive P |

Returned Value

| Reg | Description |
|---|---|
| AL = 0 | if successful |
| | -1 if disk write-protected or has files open |

Explanation

The Reorganize Disk Directory function reorganizes the directory on the disk drive specified in register DL. If the hashed-directory flag bit in the volume label has been changed, this function will convert a hashed directory into linear format (or vice versa). The principal purpose of this function is to support the FIXDIR command.

NOTE: In certain cases, this function may take a very long time to complete (possibly hours), and cannot be interrupted once invoked.

## TASM ASSEMBLER

The TASM assembler is a two-pass relocatable assembler for 8086-family microprocessors, intended for use in conjunction with the TurboDOS linker (TLINK).

## Operating Instructions

The assembler is invoked with the following command:

```
TASM sourcefn {objectfn} {-options}
```

The "sourcefn" argument identifies an ASCII text file containing one or more assembly language source modules. If "sourcefn" does not contain an explicit type, the default type .A is assumed.

The "objectfn" argument specifies the name of the object file to be created by TASM in the relocatable format required by TLINK. If "objectfn" does not contain an explicit type, the default type .O is used. If "objectfn" is omitted from the command altogether, the object file is given the same name as the source file except that type .O is used.

Options are always preceded by a "-" prefix, and may appear before, between, or after the file names. Several options may be concatenated after a single "-" prefix.

| Option | Explanation |
|---|---|
| -C | List to console, not to printer |
| -E | Allow archaic equates "=", "=:" |
| -L | Listing only, no object file |
| -S | Produce sorted symbol table |
| -U | Produce unsorted symbol table |
| -X | List only source lines in error |
| -1 | Allow 80186 instructions |

## Lexical Conventions

**Names**

A name is composed of upper case letters A-Z, lower case letters a-z, digits 0-9, and the underscore "_" character. The first character of a name may not be a digit. Upper and lower case letters are treated as different characters. Names may be of any length, but only the first eight characters are significant.

**Keywords**

The size specifiers BYTE and WORD, and the machine registers AL, BL, CL, DL, AH, BH, CH, DH, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, SS, and ES are reserved as keywords, and may not be used otherwise. Keywords may be spelled in upper or lower case.

**Location Counter**

The special symbol period "." represents the location counter value at the start of the current instruction, and may be used in expressions wherever a name would be appropriate. For example:

```
        JMP  .     ;an infinite loop
```

**Numeric Constants**

Only integer constants up to a significance of 16 bits are permitted. A sequence of digits is normally interpreted as an unsigned decimal constant. However, a sequence of digits with a leading 0 is taken to be an octal constant (in which the digits 8 and 9 are invalid). A sequence of digits preceded by 0x (or 0X) is taken to be a hexadecimal constant. Hexadecimal digits include the digits 0-9 and the letters A-F (which may be upper or lower case).

Character Constants

A character constant is a single ASCII character enclosed in apostrophes, as in 'x'. The value of a character constant is the numerical value of the character expressed in seven-bit ASCII code. Certain non-graphic characters, the apostrophe and the backslant may be represented in character constants according to the following table of escape sequences:

| ASCII Character | Representation |
|---|---|
| line feed | \n |
| horizontal tab | \t |
| back space | \b |
| carriage return | \r |
| form feed | \f |
| apostrophe | \' |
| backslant | \\ |
| null character | \0 |
| any octal code | \ddd |

The escape sequence \ddd consists of the backslant followed by 1, 2 or 3 octal digits that specify the ASCII code value of the desired character.

Strings

A string is a sequence of characters surrounded by quotes, as in "string". In a string, the quote character may be represented by the escape sequence \" and all of the escape sequences described for character constants may be used as well. No implicit string terminator is implied; if a null-terminated string is desired, it must be written as "string\0".

| | |
|---|---|
| White Space | White space (spaces and tabs) may be used freely between tokens, but not within names, keywords or constants. White space is required to separate adjacent names, keywords or constants that are not separated by punctuation (for instance, between an instruction and its operands).<br><br>Blank lines are always ignored, and may be used freely anywhere. |
| Comments | Comments are introduced by a semicolon ";" and continue until the end of the line. |

## Expressions

An expression is a sequence of names, constants, operators and parentheses that can be evaluated to yield a value. The order of evaluation is determined by the precedence and associativity of operators, unless explicitly overridden with parentheses.

### Unary Operators

The following unary prefix operators are permitted in expressions:

| Operator | Explanation |
|---|---|
| - | two's complement (negate) |
| ~ | one's complement |
| ! | logical not |

The unary operators have higher precedence than any binary operator, and are evaluated right-to-left. (For example, -~0 yields 1, while ~-0 yields -1.)

The logical not operator "!" yields a result of 1 (true) if its operand is false (zero), and a result of 0 (false) if its operand is true (nonzero).

Binary Operators

The following binary infix operators are permitted in expressions:

| Operator | Explanation |
|---|---|
| * | multiply |
| / | divide |
| % | modulus |
| + | add |
| - | subtract |
| >> | shift right |
| << | shift left |
| < | logical less-than |
| > | logical greater-than |
| == | logical equal-to |
| & | bitwise and |
| ^ | bitwise exclusive-or |
| \| | bitwise inclusive-or |
| && | logical and |
| \|\| | logical or |

Logical operators yield a result of 1 (true) or 0 (false). The logical connectives && and || treat their operands as true (if nonzero) or false (if zero).

The precedence of binary operators is shown below, with each line representing a lower precedence than the line above it:

| | | | |
|---|---|---|---|
| highest | * | / | % |
| : | + | - | |
| : | >> | << | |
| : | < | > | == |
| : | & | ^ | \| |
| lowest | && | \|\| | |

Binary operators of equal precedence are evaluated left-to-right. (For example, 5-4+3 yields 4.)

Relocatable Expressions

All operators other than add (+) and subtract (-) require absolute (non-relocatable) operands and yield an absolute result.

The addition operator (+) may be used to add a relocatable operand to an absolute operand, yielding a relocatable result with the same relocation base as the relocatable operand.

The subtraction operator (-) may be used to subtract an absolute operand from a relocatable operand, yielding a relocatable result with the same relocation base as the relocatable operand. Further, the subtraction operator may be used to take the difference between two relocatable operands with the same relocation base, yielding an absolute result.

External Expressions

A name may be declared to the assembler as external (defined in some other module) by appending the suffix "#" at the end of each reference to the name. Such an external name reference is a relocatable value. The rules for addition and subtraction of relocatable values apply to externals as well:

```
BUFFER#-1     ;valid: rel-abs
BUFFER#+0x10  ;valid: rel+abs
BEG#+LEN#     ;invalid: rel+rel
END#-BEG#     ;invalid: rel-rel
```

The last case above is invalid because each different external name is treated by the assembler as a different relocation base.

## Statements

An assembler statement consists of the following elements in the specified order:

1. a label
2. one or more instruction prefixes
3. an instruction or assignment
4. one or more operands
5. a comment

All of these elements are optional, although items 2 and 4 must be omitted if item 3 is omitted. A label must be followed by a colon ":" or an assignment operator. Multiple operands must be separated by commas. A comment must be introduced by a semicolon.

## Labels

A statement may start with a label, which consists of a name followed by a colon ":", a double-colon "::", or an assignment operator. A double-colon indicates that the label is public (may be referenced by other modules). The label is normally given the current value of the location counter (exception: the label on an assignment or EQU statement).

If a label has two leading underscore characters, such as "__LP:", it is considered to be a local label with scope limited by the preceding and following non-local labels. This allows the same local label to be re-used many times within a module without ambiguity or conflict.

Assignments

A name may be assigned any desired value by using a double-equals "==" as an assignment operator:

```
TRUE    ==      1
CR      ==      BYTE 0x0D
VAR     ==      WORD -4[BP]
```

or (equivalently) by using the EQU pseudo-instruction with a label:

```
TRUE:   EQU     1
CR:     EQU     BYTE 0x0D
VAR:    EQU     WORD -4[BP]
```

The defining expression (at the right of the assignment or EQU) may be absolute, relocatable or external, but may not contain any forward references. Note that the name is assigned the size (BYTE or WORD) and addressing mode (indexed or immediate) as well as the value and relocation characteristics of the defining expression. So, for example:

```
TABLE   ==      BYTE -8[BP]

        MOV     TABLE[SI],=0
        MOV     BYTE -8[BP+SI],=0
```

The two MOV instructions in the above example are identical.

Assignments (Continued)

To make an assignment public, use the assignment operator "==:", or use a double-colon label with the EQU pseudo-instruction:

```
LENGTH    ==:   66    ;LENGTH is public
WIDTH::   EQU   132   ;WIDTH is public
```

NOTE: Invoking TASM with the "-E" option causes the assembler to accept the archaic assignment operators "=" and "=:" as synonyms for "==" and "==:". These archaic forms are not recommended, however, because of syntactic ambiguities with the use of "=" as the immediate-addressing operator.

Prompted Assignments

An assignment statement with a string operand causes the assembler to display the given string as a prompt (followed by a colon and a space) and to accept a new operand from the console (or do-file) when the statement is encountered during the assembler's pass one:

```
DEBUG   ==    "Debug code? (0=no, 1=yes)"
BUFSIZ  ==:   "Number of buffers (1-16)"
DRIVE:  EQU   "Drive letter ('A'...'P')"
VERS::  EQU   "Enter version number"
```

In response to such a prompt, the assembler will accept any valid expression that would be legal in a non-prompted assignment. If the expression entered is not valid, the assembler asks that it be re-entered.

## Instruction Prefixes

A machine instruction may be preceded by a segment-override prefix (CS, DS, SS or ES), a repeat prefix (REP, REPE, REPZ, REPNE or REPNZ), the prefix LOCK, or a combination of these. For example:

```
    REP MOVS BYTE        ;repeat until CX=0
    LOCK CS MOV  BX,RTN ;RTN in CS-segment
```

Alternatively, each prefix may appear as a separate statement:

```
          LOCK              ;lock prefix
          CS                ;seg-override
          MOV     BX,RTN    ;prefixed instr.
```

Note that segment-override prefixes must always be given explicitly, as the assembler never generates them implicitly.

## Instructions

The instruction part of a statement may be either a symbolic machine instruction or a pseudo-instruction. Instructions and pseudo-instructions may be spelled in upper or lower case.

Addressing Modes

Expressions are presumed to represent direct memory addresses unless immediate or indexed addressing is explicitly indicated. The equals-sign "=" may be used as a prefix to indicate that an immediate value is intended:

```
MOV  AX,=0x1000   ;loads value 0x1000
MOV  AX,0x1000    ;loads word at DS:1000
```

If an address is intended to be used as an immediate operand, the ampersand "&" prefix may be used to indicate "address of":

```
MOV  AX,&BUFFER   ;loads addr of BUFFER
MOV  AX,BUFFER    ;loads word at BUFFER
```

Actually, the effect of the immediate-addressing prefixes "=" and "&" is identical except that the "&" prefix discards any size-attribute which the prefixed expression may have, while the "=" prefix does not.

An indexing expression enclosed in brackets "[...]" may be used to indicate that an indexed addressing mode is intended:

```
MOV  AX,[BX]          ;addr=(BX)
MOV  AX,[BX+SI]       ;addr=(BX)+(SI)
MOV  AX,-4[BP]        ;addr=(BP)-4
MOV  AX,BUFFER[BX]    ;addr=BUFFER+(BX)
```

Operand Size

Many 8086-family data manipulation instructions can operate on either bytes or words. For most such instructions, the assembler can determine implicitly whether to generate a byte or word instrution:

```
MOV   AX,=0   ;word (AX is word-length)
MOV   AL,=0   ;byte (AL is byte-length)
PUSH  VALUE   ;word (can't PUSH a byte)
```

However, it is necessary to specify the operand length explicitly with the keyword BYTE or WORD if the assembler cannot otherwise determine it:

```
MOV  BYTE -4[BP],=0      ;byte
MOV  WORD -4[BP],=0      ;word
MOV  -4[BP],WORD =0      ;same as above
MOV  -4[BP],=WORD 0      ;ditto
MOV  DL,WORD -4[BP]      ;invalid
```

The last example is invalid because the size attributes of the source (WORD) and destination (BYTE) operands clash.

## Pseudo-Instructions

Module Identification

The pseudo-instruction MODULE defines the module identification that appears at the top of each TASM listing page and in the in the TLINK module map. The module identification must be enclosed in quotes:

```
MODULE  "MAINPROG"    ;module ident
```

and is truncated to 8 characters if a longer identification is specified.

Linker Control

The pseudo-instruction TLINK specifies one or more TLINK option letters enclosed in quotes:

```
TLINK  "HX"  ;force TLINK -H and -X opt
```

and causes those options to be in effect whenever the module is processed by TLINK.

Location Counter

The pseudo-instruction LOC (or ORG) sets the assembler's location counter to the value of its operand. The operand may be any valid absolute, relocatable or external expression, but it must not include forward references:

```
        LOC     0x100   ;absolute
        LOC     Code#   ;external
```

If the operand is absolute, the assembler will assign absolute addresses to the code and data statements which follow, starting with the given absolute address.

Location Counter (Continued)

If the operand is relocatable (generally an external name reference), the assembler will assign relocatable addresses relative to the relocation base of the operand. A relocation base may be any external name, but the following special names are recognized by TLINK:

```
        LOC     Code#   ;code segment
        LOC     Data#   ;data segment
        LOC     Extra#  ;extra segment
        LOC     Stack#  ;stack segment
```

Note the initial upper-case letter followed by lower-case letters (remember, case is significant in TASM names).

The pseudo-instruction RELOC (or REORG) restores the location counter to the value it had just prior to the preceding LOC (or ORG):

```
        LOC     Code#   ;code segment
START:  MOV     BX,TABLE
        MOV     CX,=TABLEN
__L:    CALL    SUBRTN
        INC     BX
        LOOP    __L
        RETF

        LOC     Data#   ;data segment
TABLE:  BYTE    3,5,7,11,13,17
TABLEN  ==      .-TABLE

        RELOC           ;code seg again
SUBRTN: ...
        RET
```

Note above the "RELOC" statement could be replaced by a second "LOC Code#" (entirely equivalent).

Data Definition

The pseudo-instruction BYTE (or DB) generates one or more byte-length data values:

```
BYTDAT: BYTE    ZZZ, 4*X, WORD ALPHA
        BYTE    "Hello\r\n\0"
```

The label (if present) is given the BYTE size-attribute. A string operand generates one byte for each character in the string. An operand with an explicit size-attribute WORD generates a word of data. All other operands generate a byte of data.

The pseudo-instruction WORD (or DW) generates one or more word-length data values:

```
WRDDAT: WORD    ALPHA, 234*BETA, BYTE 5
        WORD    "What's the good word\0"
```

The label (if present) is given the WORD size-attribute. A string operand generates one byte for each character in the string. An operand with an explicit size-attribute BYTE generates a byte of data. All other operands generate a word of data.

The pseudo-instruction RES (or RS) causes a specified number of bytes or words to be reserved without initialization:

```
BLOCK:  RES  0x100   ;reserve 256 bytes
BBLOCK: RES  BYTE 64 ;reserve 64 bytes
WBLOCK: RES  WORD 64 ;reserve 64 words
```

Data Definition (Continued)

If the operand of a labelled RES statement has an explicit size-attribute (BYTE or WORD), then the label is given the same size-attribute. If the operand has an explicit WORD size-attribute, then the statement reserves the specified number of words; otherwise, it reserves the specified number of bytes.

The pseudo-instruction ALIGN causes the next generated item to be word-aligned (that is, assigned an even-numbered address):

```
          ALIGN
WRDDAT:   WORD    GAMMA    ;word-aligned
          BYTE    OMEGA
          ALIGN
          RES     WORD 48  ;word-aligned
STACK     ==      .
```

ALIGN is most frequently used before WORD or RES statements.

End of Module

The pseudo-instruction END terminates a module, and may have an optional operand that specifies a program starting address:

```
          MODULE  "ALPHA"
START:    ...
          ...
          END     START
```

An assembler source file may contain multiple modules. Each module is terminated with an END statement. The END statement following the last module in the file is optional (but recommended).

**Pre-Processor Directives**

Pre-processor directives differ from statements in that (1) they always start with a number-sign "#" prefix, (2) they may not have a label, and (3) they do not appear in the assembler listing.

Listing Control

The #NOLIST directive prevents succeeding statements from appearing in the assembler listing. The #LIST directive re-enables listing after a #NOLIST.

The #RELIST directive restores the listing mode that was in effect just prior to the last #LIST or #NOLIST directive. Nesting is not permitted.

Listing Format

The #PAGE directive may take three forms:

```
#PAGE  width,length     ;set width+length
#PAGE  width            ;set width only
#PAGE                   ;start a new page
```

The first two forms change the page width and length used for the assembler listing from their default values of 80 columns/line and 66 lines/page. The last form (with no operands) forces the start of a new listing page.

The following directives:

```
#TITLE  "Title of this module"
#SUBTTL "Sub-title of this module"
```

cause the specified strings to be used as a title or subtitle at the top of each page of the assembler listing.

File Inclusion

A directive of the form:

```
#INCLUDE  "filename"
```

causes the entire contents of the specified source file to be included at that point in the source program. The file name must be enclosed in quotes. If no file type is specified, the default type .A is assumed. If no drive is specified, then the drive of the original "srcefile" argument from the TASM command line is assumed. #INCLUDE directives may be nested.

Conditional Assembly

Conditional assembly is achieved by using the following directives:

```
#IF     expression
...
#ELSE
...
#ENDIF
```

The #IF-expression must yield an absolute value and must not contain forward references. If the expression evaluates to true (nonzero), any lines between the #ELSE and the #ENDIF are ignored by the assembler. If the expression evaluates to false (zero), any lines between the #IF and the #ELSE (or the #ENDIF if there is no #ELSE) are ignored. #IF-#ELSE-#ENDIF sequences may be nested.

Repetition

The directives:

```
#REPEAT expression
...
#ENDREP
```

cause any lines between the #REPEAT and #ENDREP directives to be repeated the number of times specified by the #REPEAT-expression. The expression must not contain forward references, and must evaluate to an absolute positive value between 1 and 32,767. Otherwise, the #REPEAT directive is diagnosed and a repeat-factor of one is assumed by the assembler. #REPEAT-#ENDREP may be nested.

Macro Definition

TASM does not support macros (yet).

## Machine Instructions

This section lists all machine instructions known to the assembler, together with the types of operands they require. Instructions marked "*" are 80186 instructions, and are diagnosed by the assembler unless the "-1" option is specified.

| Instr. | Operands | Explanation |
|---|---|---|
| AAA | | ASCII adj add |
| AAD | | ASCII adj div |
| AAM | | ASCII adj mult |
| AAS | | ASCII adj subtr |
| ADC | reg,reg/mem | add with carry |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| ADD | reg,reg/mem | add |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| AND | reg,reg/mem | and logical |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| BOUND* | reg,mem | bounds check |
| CALL | label | call near |
| CALLF | label,para | call far |
| CALLFI | reg/mem | call far indir |
| CALLI | reg/mem | call near indir |
| CALLIF | reg/mem | call far indir |
| CBW | | convert byte/wd |
| CLC | | clear carry |
| CLD | | clear direction |
| CLI | | clear interrupt |
| CMC | | complemnt carry |
| CMP | reg,reg/mem | compare |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| CMPS | BYTE/WORD | compare string |
| CWD | | convert wd/dblw |
| DAA | | decimal adj add |
| DAS | | decimal adj sub |
| DEC | reg/mem | decrement |
| DIV | reg/mem | divide |

Machine Instructions (Continued)

| Instr. | Operands | Explanation |
|---|---|---|
| ENTER* | =frame,=nest | enter procedure |
| ESC | const,reg/mem | escape |
| HLT | | halt |
| IDIV | reg/mem | integer divide |
| IMUL | reg/mem | integer multply |
| | reg,=immed* | |
| | reg,reg/mem,=immed* | |
| IN | accum,const | input |
| | accum,DX | |
| INS* | BYTE/WORD,DX | input string |
| INC | reg/mem | increment |
| INT | const | interrupt |
| INTO | | interrupt o'flo |
| IRET | | interrupt ret'n |
| JA | label | jump if above |
| JAE | label | jump if abv/eq |
| JB | label | jump if below |
| JBE | label | jump if blo/eq |
| JC | label | jump if carry |
| JCXZ | label | jump if CX=0 |
| JE | label | jump if equal |
| JG | label | jump if greater |
| JGE | label | jump if grtr/eq |
| JL | label | jump if less |
| JLE | label | jump if less/eq |
| JMP | label | jump near |
| JMPF | label,para | jump far |
| JMPFI | reg/mem | jump far indir |
| JMPI | reg/mem | jump near indir |
| JMPIF | reg/mem | jump far indir |
| JMPS | label | jump short |
| JNA | label | jump not above |
| JNAE | label | jump not abv/eq |
| JNB | label | jump not below |
| JNBE | label | jump not blo/eq |
| JNC | label | jump not carry |
| JNE | label | jump not equal |
| JNG | label | jump not greatr |
| JNGE | label | jump not gtr/eq |

**Machine Instructions (Continued)**

| Instr. | Operands | Explanation |
|---|---|---|
| JNL | label | jump not less |
| JNLE | label | jump not les/eq |
| JNO | label | jump not o'flo |
| JNP | label | jump not parity |
| JNS | label | jump not sign |
| JNZ | label | jump not zero |
| JO | label | jump if o'flo |
| JP | label | jump if parity |
| JPE | label | jump if pty evn |
| JPO | label | jump if pty odd |
| JS | label | jump if sign |
| JZ | label | jump if zero |
| LAHF | | load AH=flags |
| LDS | reg,reg/mem | load ptr w/DS |
| LEA | reg,reg/mem | load efctv addr |
| LEAVE* | | leave procedure |
| LES | reg,reg/mem | load ptr w/ES |
| LOCK | | lock prefix |
| LODS | BYTE/WORD | load string |
| LOOP | label | loop |
| LOOPE | label | loop while eq |
| LOOPNE | label | loop while neq |
| LOOPNZ | label | loop while nonz |
| LOOPZ | label | loop while zero |
| MOV | reg,reg/mem | move |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| | seg,reg/mem | |
| | reg/mem,seg | |
| MOVS | BYTE/WORD | move string |
| MUL | reg/mem | multiply |
| NEG | reg/mem | negate |
| NOP | | no operation |
| NOT | reg/mem | logical not |
| OR | reg,reg/mem | logical or |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| OUT | const,accum | output |
| | DX,accum | |
| OUTS* | DX,BYTE/WORD | output string |

Machine Instructions (Continued)

| Instr. | Operands | Explanation |
|---|---|---|
| POP | reg/mem | pop |
| | seg | |
| POPA* | | pop all |
| POPF | | pop flags |
| PUSH | reg/mem | push |
| | seg | |
| | =immed* | |
| PUSHA* | | push all |
| PUSHF | | push flags |
| RCL | reg/mem,=1 | rotate cy left |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| RCR | reg/mem,=1 | rotate cy right |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| REP | | repeat |
| REPE | | repeat while eq |
| REPNE | | repeat while ne |
| REPNZ | | repeat while nz |
| REPZ | | repeat while z |
| RET | | return near |
| | const | |
| RETF | | return far |
| | const | |
| ROL | reg/mem,=1 | rotate left |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| ROR | reg/mem,=1 | rotate right |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| SAHF | | store AH=>flags |
| SAL | reg/mem,=1 | shift ar left |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| SAR | reg/mem,=1 | shift ar right |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| SBB | reg,reg/mem | subtract borrow |
| | reg/mem,reg | |
| | reg/mem,=immed | |

**Machine Instructions (Continued)**

| Instr. | Operands | Explanation |
|---|---|---|
| SCAS | BYTE/WORD | scan string |
| SHL | reg/mem,=1 | shift left |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| SHR | reg/mem,=1 | shift right |
| | reg/mem,CL | |
| | reg/mem,=immed* | |
| STC | | set carry |
| STD | | set direction |
| STI | | set interrupt |
| STOS | BYTE/WORD | store string |
| SUB | reg,reg/mem | subtract |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| TEST | reg,reg/mem | test |
| | reg/mem,reg | |
| | reg/mem,=immed | |
| WAIT | | wait |
| XCHG | reg,reg/mem | exchange |
| | reg/mem,reg | |
| XLAT | | translate |
| XOR | reg,reg/mem | exclusive or |
| | reg/mem,reg | |
| | reg/mem,=immed | |

## TLINK LINKER

TLINK is a specialized linker used for 8086 TurboDOS system generation, and may also be used as a general-purpose linker for object modules produced by the TASM assembler. TLINK links a specified collection of object modules together into a single executable file.

## Operation

The linker is invoked with the following command:

```
TLINK inputfn {outputfn} {-options}
```

The "inputfn" argument identifies the two input files used by the linker: a configuration file "inputfn.GEN" and a parameter file "inputfn.PAR". The "outputfn" argument specifies the name of the executable output file to be created (normally type .CMD or .SYS). If "outputfn" is omitted from the command, then "inputfn" is also used as the name of the executable output file, and should include an explicit file type (.CMD or .SYS).

If the .GEN file is found, it must contain the list of object modules (.O files) to be linked together. If the configuration file is not found, then TLINK operates in an interactive mode. You are prompted by an asterisk * to enter a series of directives from the console. The syntax of each directive (or each line of the .GEN file) is:

```
objfile {,objfile}... {;comment}
```

Operation (Continued)

The object files are assumed to have type .O unless a type is given explicitly. A null directive (or the end of the .GEN file) terminates the prompting sequence and causes processing to proceed.

After obtaining the list of modules from the file or console, TLINK links all of the modules together, a two-pass process that displays the name of each module as it is encountered. When the linking phase is complete, TLINK looks for a parameter file "inputfn.PAR" and processes it if present (described below). Finally, the executable file (.CMD or .SYS) is written out to disk.

NOTE: Each module of the TurboDOS operating system is magnetically serialized with a unique serial number. The serial number consists of two components: an "origin number" which identifies the issuing TurboDOS licensee, and a "unit number" which uniquely identifies each copy of TurboDOS issued by that licensee. When used for TurboDOS operating system generation, TLINK verifies that all modules to be linked are serialized consistently, and serializes the executable file accordingly.

Options

Options are always preceded by a "-" prefix, and may appear before, between, or after the file names. Several options may be concatenated after a single "-" prefix.

| Option | Explanation |
|---|---|
| -8 | Force 8080 model (single group) |
| -B | No 128-byte base page |
| -C | List to console, not to printer |
| -D | Force data group G-Max to 64K |
| -H | No .CMD header (implies -8, -B) |
| -L | Listing only, no output file |
| -M | List link map |
| -R | List inter-module references |
| -S | List sorted symbol table |
| -U | List unsorted symbol table |
| -X | Diagnose undefined references |

Parameter File

TLINK includes a symbolic patch facility that may be used during TurboDOS system generation to override various operating system parameters and to effect necessary software corrections. Symbolic patches must be stored in a .PAR file which may be built using any text editor. The syntax of each .PAR file entry is:

```
location = value {,value}... {;comment}
```

where the "value" arguments are to be stored in consecutive memory locations starting with the address specified by "location".

Parameter File (Continued)

The "location" argument may be the name of a public symbol, an integer constant, or an expression composed of names and integer constants connected by + or - operators. Integer constants must begin with a digit to distinguish them from names. Constants of the form "0xdddd" are taken to be hexadecimal. Constants of the form "0ddddddd" are taken to be octal. Constants that start with a nonzero digit are taken to be decimal. The "location" expression must be followed by an equal-sign = character.

The "value" arguments may be expressions (as defined above) or quoted ASCII strings, and must be separated by commas. A "value" expression is stored as a 16-bit word if its value exceeds 255 or if it is enclosed in parentheses; otherwise, it is stored as an 8-bit byte. A quoted ASCII string must be enclosed by quotes "...", and is stored as a sequence of 8-bit bytes. Within a quoted string, ASCII control characters may be specified by using backslant escape sequences (as described in the section on TASM).

Error Messages

```
Serial number violation
Not enough memory
No object files specified
Can't open object file
Unexpected EOF in object file
Bad token in object file: <type>
Can't create output file
Can't write output file
Load address out-of-bounds
Duplicate transfer address
Duplicate def: <name>
Undefined name: <name>
Too many externals in module
Name table overflow
```

## TBUG DEBUGGER

TBUG is an interactive debuging facility that provides various facilities under 8086 TurboDOS useful to programmers who have the need to debug or patch programs.

## Operation

The debugger is invoked with one of the following commands:

```
TBUG
TBUG filename
TBUG "filename commandtail"
```

The first form simply invokes the debugger. The second form also causes the specified program file to be loaded into memory (see the L-directive below); the named file must have a .CMD header. The third form loads the specified program and parses the given command tail (see the Z-directive below); in this form the enclosing quotes are required.

TBUG operates in an interactive mode. You are prompted by an asterisk * to enter a series of directives from the console. The Q-directive (Quit) terminates TBUG.

Following is a summary of TBUG directives:

```
A - display memory in ASCII
C - calculate hexadecimal sum/difference
D - display memory in hexadecimal
E - examine/alter memory contents
F - fill memory block with constant value
G - start execution, set breakpoints
H - display "help" menu of directives
I - input from specified input port
L - load program from .CMD  file
M - move a memory block
```

**Operation (Continued)**

```
O - output to specified output port
P - put ASCII text into memory
Q - quit TBUG and return to TurboDOS
S - save program to .CMD file
T - trace in single-instruction mode
U - un-assemble code into TASM mnemonics
V - verify if two memory blocks are equal
W - breakpoint on specified OS calls
X - examine/modify machine registers
Z - parse command line into base page
```

**Directive Syntax**

Each TBUG directive starts with a letter which specifies the action to be taken, and ends with a carriage return. The directive letter may be followed by one or more arguments (addresses, address ranges, values, file names, etc.) separated by commas or spaces.

Memory Addresses

Most TBUG directives require one or more memory addresses as arguments. Addresses may be entered in three alternative formats:

```
BBBB:0000   base paragraph + offset
RR:0000     segment register + offset
0000        offset only
```

The first format consists of a hexadecimal segment base paragraph address BBBB plus a hexadecimal offset byte address 0000. In the second format, the segment base is specified by naming one of the 8086-family segment registers CS, DS, ES or SS. In the third format, the segment base is not specified; TBUG assumes the base is CS for the G, T and U directives, and DS for other directives.

Address Ranges

Some TBUG directives accept a memory address range as an argument. Address ranges may be entered in two alternative formats:

```
startaddr,endaddr
startaddr,Llength
```

The first format specifies the range as a starting address and ending address, separated by a comma (or a space). The starting address may contain a segment base prefix (paragraph address or segment register name), but the ending address must not (it is assumed to have the same segment base as the starting address).

The second format specifies the range as a starting address and a length (in hexadecimal bytes). The length must be prefixed with the letter "L" to indicate that it is a length rather than an ending address.

**Directives**

A-Directive

The A-directive displays the contents of a block of memory in ASCII. The directive formats are:

```
A
A address
A range
```

The first format displays 128 bytes of memory starting from the last address previously displayed. The second format displays 128 bytes of memory starting from the given address. The third format displays the given address range.

C-Directive

The C-directive displays the sum and difference of two hexadecimal arguments. The directive format is:

```
C  value1  value2
```

in response to which TBUG displays the hexadecimal sum and difference of the two arguments.

D-Directive

The D-directive displays the contents of a block of memory in hexadecimal. The directive formats are:

```
D
D  address
D  range
```

The first format displays 128 bytes of memory starting from the last address previously displayed. The second format displays 128 bytes of memory starting from the given address. The third format displays the given address range.

E-Directive

The E-directive is used to examine and modify the contents of memory. The directive format is:

```
E  address
```

TBUG displays the hexadecimal byte at the given address followed by an equals sign = and awaits keyboard input.

E-Directive (Continued)

If a hexadecimal value is entered, it is stored at that memory location. If an equals sign = is entered, the memory location is left unchanged. In either case, TBUG continues to display successive memory addresses and values until a null response (RETURN only) is entered.

F-Directive

The F-directive fills a block of memory with zeroes, or with a specified hexadecimal byte value. The directive formats are:

```
F  range
F  range  value
```

The first form fills every location in the given address range with zero. The second form fills every location in the range with the given byte value.

G-Directive

The G-directive starts executing the loaded program, and optionally sets one or more breakpoint addresses. The directive formats are:

```
G
G  =address
G  breakpoint...
G  =address  breakpoint...
```

The first format transfers to the starting address corresponding to the current values of the CS and IP registers. The second format transfers to the given starting address, setting the CS and IP registers accordingly.

G-Directive (Continued)

The last two formats are similar to the first two, except that up to ten breakpoint addresses are specified. If the program encounters any of the breakpoints, execution is interrupted just prior to the instruction at the breakpoint address, the address is displayed, all outstanding breakpoints are cancelled, and TBUG prompts for another directive.

H-Directive

The H-directive displays a help menu that lists all TBUG directives, each with its argument format and a brief description.

I-Directive

The I-directive inputs a byte from an input port. The directive format is:

```
I port
```

where "port" is a hexadecimal input port address. A byte is input from the specified port and displayed in hexadecimal.

L-Directive

The L-directive loads a program into memory from disk. The directive format is:

```
L filename {commandtail}
```

If "filename" does not specify an explicit type, the default type .CMD is assumed. In any case, the file must start with a .CMD header. TBUG discards any previously loaded program, loads the specified .CMD file into memory, and initializes the base page, segment registers and IP register. If a command tail is present, it is parsed and processed as described under "Z-Directive" below.

M-Directive

The M-directive moves a block of memory to another location. The directive format is:

```
M  range  address
```

The block of memory specified by "range" is moved to the starting address specified by "address".

O-Directive

The O-directive outputs a specified byte value to a specified output port. The directive format is:

```
O  port  value
```

where "port" is a hexadecimal output port address and "value" is a hexadecimal byte value. The given value is output to the given port.

P-Directive

The P-directive permits ASCII text to be entered from the console into memory. The directive format is:

```
P  address
```

In response to this directive, TBUG accepts console input and stores each ASCII character into a successive memory location, starting at the given address. Entering an EOT character (CTRL-D) terminates the directive.

Q-Directive

The Q-directive is used to quit TBUG and return to TurboDOS.

S-Directive

The S-directive saves the currently loaded program onto disk. The directive format is:

```
    S  filename
```

If "filename" does not specify an explicit type, the default type .CMD is assumed. The currently loaded program is saved on disk in .CMD format under the specified file name.

Note that whenever TBUG loads a program into memory, it retains information about the segment structure of the loaded program. The S-directive uses this information to determine the program segment structure to be written to disk.

T-Directive

The T-directive traces program execution in single-instruction mode. The directive formats are:

```
    T
    T  =address
    T  length
    T  =address  length
```

The first format traces the instruction corresponding to the current values of the CS and IP registers. The second format traces the instruction at the given starting address, setting the CS and IP registers accordingly. The last two formats are similar to the first two, except that "length" specifies the hexadecimal number of instructions to be traced.

U-Directive

The U-directive displays the contents of memory "un-assembled" into TASM mnemonics. The directive formats are:

```
U
U  =address
U  length
U  =address  length
```

The first format displays the next 16 machine instructions, starting from the last address previously displayed. The second format displays the next 16 machine instructions, starting from the specified address. The last two formats are similar to the first two, except that "length" specifies the hexadecimal number of instructions to be displayed.

V-Directive

The V-directive verifies whether or not two blocks of memory are identical. The directive format is:

```
V  range  address
```

The block of memory specified by "range" is compared to the block of equal length starting at "address". Any discrepancies are diagnosed.

W-Directive

The W-directive executes the loaded program in monitored mode, breaking on specified C- and T-function calls. The format is:

```
W  fcn...
```

where up to ten "fcn" arguments may be specified to trap specific TurboDOS function calls. Each "fcn" argument may take one of the following forms:

```
nn      (trap C-function nn hex)
Tnn     (trap T-function nn hex)
*       (trap all C-functions)
T*      (trap all T-functions)
```

Program execution starts at the location specified by the current CS and IP register values, and continues until one of the trapped functions is invoked by the program. Program termination is always trapped.

X-Directive

The X-directive is used to display and alter the contents of machine registers. The directive formats are:

```
X
X  regname
```

The fist format displays the contents of all machine registers. The second format displays the contents of the specified register, and permits it to be altered by entering a hexadecimal value. Only word-length register names are accepted: AX, BX, CX, DX, SI, DI, BP, SP, IP, CS, DS, ES and SS.

Z-Directive

The Z-directive sets up the default FCB and default record buffer in the base page of the currently loaded program according to the given command-tail parameters. The command format is:

```
Z  command-tail
```

The command tail length and text are moved to the base page record buffer, and up to two filenames are parsed from the command tail and placed into the base page FCB.

| CL= | C-Function Name | Arguments Passed | Values Returned |
|---|---|---|---|
| 0 | System Reset | - | - |
| 1 | Console Input | - | AL = char |
| 2 | Console Output | DL = char | - |
| 3 | Raw Console Input | - | AL = char |
| 4 | Raw Console Output | DL = char | - |
| 5 | List Output | DL = char | - |
| 6 | Direct Console I/O | DL = -1 (inp/sta) | AL = 0/char |
| | | DL = -2 (status) | AL = 0/-1 |
| | | DL = -3 (input) | AL = char |
| | | DL = char (output) | - |
| 7 | Get I/O Byte | - | AL = I/O byte |
| 8 | Set I/O Byte | DL = I/O byte | - |
| 9 | Print String | DS:DX = &string | - |
| 10 | Read Console Buffer | DS:DX = &buffer | - |
| 11 | Get Console Status | - | AL = 0/-1 |
| 12 | Return Version | - | BH = 0 |
| | | | BL = 31H |
| 13 | Reset Disk System | - | - |
| 14 | Select Disk | DL = drive (0=A) | - |
| 15 | Open File | DS:DX = &FCB | AL = (-1 if err) |
| 16 | Close File | DS:DX = &FCB | AL = (-1 if err) |
| 17 | Search for First | DS:DX = &FCB | AL = (-1 if err) |
| 18 | Search for Next | - | AL = (-1 if err) |
| 19 | Delete File | DS:DX = &FCB | AL = (-1 if err) |
| 20 | Read Sequential | DS:DX = &FCB | AL = (NZ if err) |
| 21 | Write Sequential | DS:DX = &FCB | AL = (NZ if err) |
| 22 | Make File | DS:DX = &FCB | AL = (-1 if err) |
| 23 | Rename File | DS:DX = &FCB | AL = (-1 if err) |
| 24 | Return Login Vector | - | BX = vector |
| 25 | Return Current Disk | - | AL = drive (0=A) |
| 26 | Set DMA Address | DS:DX = &DMA | - |
| 27 | Get ALV Address | (not supported) | BX = 0 |
| 28 | Write Protect Disk | - | - |
| 29 | Get R/O Vector | - | BX = vector |
| 30 | Set File Attributes | DS:DX = &FCB | AL = (-1 if err) |
| 31 | Get DPB Address | - | BX = &DPB |
| 32 | Get/Set User Number | DL = -1 | AL = user number |
| | | DL = user number | - |
| 33 | Read Random | DS:DX = &FCB | AL = (NZ if err) |
| 34 | Write Random | DS:DX = &FCB | AL = (NZ if err) |

| CL= | C-Function Name | Arguments Passed | Values Returned |
|---|---|---|---|
| 35 | Compute File Size | DS:DX = &FCB | AL = (-1 if err) |
| 36 | Set Random Record | DS:DX = &FCB | - |
| 37 | Reset Drive | DX = vector | - |
| 40 | Write Random 0-Fill | DS:DX = &FCB | AL = (NZ if err) |
| 42 | Lock Record | DS:DX = &FCB | AL = (NZ if err) |
| 43 | Unlock Record | DS:DX = &FCB | AL = (NZ if err) |
| 46 | Get Disk Free Space | DL = drive (0=A) | AL = 0 |
| 47 | Chain to Program | (Cmd at 0x0080) | - |
| 50 | Direct BIOS Call | DS:DX = &BIOS Desc | AX = BX = return |
| 51 | Set DMA Base | DX = DMA base para | - |
| 52 | Get DMA Address | - | ES:BX = DMA addr |
| 53 | Alloc Max Memory | DS:DX = &MCB | AL = (-1 if err) |
| 54 | Alloc Abs Max Mem | DS:DX = &MCB | AL = (-1 if err) |
| 55 | Allocate Memory | DS:DX = &MCB | AL = (-1 if err) |
| 56 | Alloc Abs Memory | DS:DX = &MCB | AL = (-1 if err) |
| 57 | Free Memory | DS:DX = &MCB | AL = (-1 if err) |
| 58 | Free All Memory | - | - |
| 59 | Program Load | DS:DX = &FCB | BX = BP para/-1 |
| 104 | Set Date and Time | DS:DX = &DTP | - |
| 105 | Get Date and Time | DS:DX = &DTP | AL = seconds/BCD |
| 107 | Return Serial Nbr | DS:DX = &SN | - |
| 108 | Get/Set Return Code | DX = 0xFFFF | BX = retcode |
| | | DX = retcode | - |
| 110 | Get/Set Delimiter | DX = 0xFFFF | AL = delimiter |
| | | DL = delimiter | - |
| 111 | Print Block | DS:DX = &CCB | - |
| 112 | List Block | DS:DX = &CCB | - |
| 152 | Parse Filename | DS:DX = &PFCB | BX = 0 if EOL<br>-1 if error<br>else &delim |

| CL= | T-Function Name | Arguments Passed | Values Returned |
|---|---|---|---|
| 0 | Reset O/S | - | - |
| 1 | Create Process | DX = &entrypoint<br>BX = &workspace | AL = 0/-1 |
| 2 | Delay Process | DX = tick count | - |
| 3 | Allocate Memory | DX = length | AL = 0/-1<br>BX = &memory |
| 4 | Deallocate Memory | DX = &memory | - |
| 5 | Send I/P Message | DX = &msgnode<br>BX = &message | - |
| 6 | Receive I/P Message | DX = &msgnode | BX = &message |
| 7 | Set Error Address | BX:DX = &errorcode | - |
| 8 | Set Abort Address | BX:DX = &abortcode | - |
| 9 | Set Date and Time | BX = Julian date<br>DH = hours<br>DL = minutes<br>CH = seconds | - |
| 10 | Get Date and Time | - | BX = Julian Date<br>DH = hours<br>DL = minutes<br>CH = seconds<br>CL = tick count |
| 11 | Rebuild Disk Map | DL = drive (A=0) | AL = 0/-1 |
| 12 | Return Serial Nbr | - | BX = origin #<br>DX = unit #<br>CH = 0x80 (priv)<br>CL = 0x13 vers'n |
| 13 | Set Compatability | DL = compatflags | - |
| 14 | Log-On/Log-Off | DX = 0FFFFH (off)<br>DH = -1/drive (on)<br>DL = user nbr (on) | AL = 0/-1 |
| 15 | Load File | DS:DX = &FCB | AL = 0/1/-1 |
| 16 | Activate Do-File | DS:DX = &FCB | AL = 0/-1 |
| 17 | Dis/Enable Autoload | DL = 0 (disable)<br>DL = 1 (enable) | - |
| 18 | Send Command Line | DSLDX = &buffer | - |
| 19 | Get Alloc Info | DL = drive (0=A) | AL = block size<br>CL = dir blocks<br>DX = free blocks<br>BX = tot. blocks |

| CL= | T-Function Name | Arguments Passed | Values Returned |
|---|---|---|---|
| 20 | Get Physical Info | DL = drive (0=A) | AL = sector size<br>CX = res. tracks<br>DX = tot. tracks<br>BX = sectors/trk |
| 21 | Get/Set Drv Status | DL = drive (0=A)<br>DH = 0 (set R/W)<br>DH = 1 (set R/O)<br>DH = -1 (get) | AL = 0/-1<br><br><br>BL = -1 if ready<br>BH = -1 if R/O |
| 22 | Phys. Disk Access | DS:DX = &PDR | AL = 0/-1 |
| 23 | Set Buffer Params | DH = # of buffers<br>DL = buffer size | - |
| 24 | Get Buffer Params | - | AL = mem. size<br>BH = # buffers<br>BL = buffer size |
| 25 | Lock/Unlock Drive | DL = drive (0=A)<br>DH = 0 (unlock)<br>DH = -1 (lock) | AL = 0/-1 |
| 26 | Flush/Free Buffers | DL = drive (0=A)<br>DH = subfunctions | - |
| 27 | Get/Set Print Mode | DL = print mode<br>DH = printer/queue<br>CH = spool drive | AL = spool drive<br>BH = prntr/queue<br>BL = print mode |
| 28 | Signal End-of-Print | - | - |
| 29 | Get/Set Despool Mod | DL = despool mode<br>DH = queue assgnmt<br>CH = printer | AL = 0/-1 |
| 30 | Queue a Print File | DS:DX = &FCB<br>BH = print queue<br>BL = user#/delete | AL = 0/-1 |
| 31 | Flush List Buffer | - | - |
| 32 | Network List Out | DL = char | - |
| 33 | Remote Console I/O | DL = 0/char<br>DH = -1 to attach | AL = 0/1/-1 |
| 34 | Get Comm Status | DH = channel/rmt | AL = 0/-1 |
| 35 | Comm Channel Input | DH = channel/rmt | AL = char |
| 36 | Comm Channel Output | DH = channel/rmt<br>DL = char | - |

| CL= | T-Function Name | Arguments Passed | Values Returned |
|---|---|---|---|
| 37 | Set Comm Baud Rate | DH = channel/rmt<br>DL = baudrate | - |
| 38 | Get Comm Baud Rate | DH = channel/rmt | AL = baudrate |
| 39 | Set Modem Controls | DH = channel/rmt<br>DL = vector | - |
| 40 | Get Modem Status | DH = channel/rmt | AL = vector |
| 41 | User-Defined Fcn | CH = net routing<br>BX & DX userdef | AX...DX userdef |
| 42 | Reorg Disk Dir | DL = drive (0=A) | AL = 0/-1 |

## INDEX