8086 Implementer's Guide

Copyright Notice

Trademark Notice

Disclaimer

First Edition: January 1984

ABOUT THIS GUIDE

Purpose

We've designed this 8086 Implementor's Guide to provide the information you need to know in order to generate various TurboDOS configurations for 8086-family microcomputers, and to write the driver modules for various peripheral devices. This document describes the modular architecture and internal programming conventions of TurboDOS, and explains the procedures for system generation, serialization, and distribution. It also provides detailed interface specifications for hardware-dependent driver modules.

Assumptions

In writing this guide, we've assumed that you are an OEM, dealer, or sophisticated TurboDOS user, knowledgable in 8086-family microcomputer hardware and assembly-language programming. We've also assumed you have read both the User's Guide and the 8086 Programmer's Guide, and are therefore familiar with the commands, external features, and internal functions of 8086 TurboDOS.

Organization

This guide starts with a section that describes the architecture of TurboDOS. It explains the function of each internal module of the operating system, and how these modules may be combined to create the various configurations of TurboDOS.

The next section explains the system generation procedure in detail, and describes each TurboDOS parameter which can be modified during system generation.

The third section of this guide explains the TurboDOS distribution procedure, including licensing, serialization, and support.

Organization (Continued)

The fourth section is devoted to an in-depth discussion of internal programming conventions, aimed at the programmer writing drivers or resident processes for TurboDOS.

The fifth section presents formal interface specifications for implementing hardware-dependent driver modules.

This guide concludes with a large appendix containing assembler source listings of actual driver modules. The sample drivers cover a wide range of peripheral devices, and provide an excellent starting point for programmers involved in driver development.

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

You should read the first two volumes before start into this document. The User's Guide introduces the external features and facilities of TurboDOS, and describes each TurboDOS command. The 8086 Programmer's Guide explains the internal workings of TurboDOS, and describes each operating system function in detail.

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

ARCHITECTURE

This section introduces you to the internal architecture of the TurboDOS operating system. TurboDOS is highly modular, consisting of more than forty separate functional modules distributed in relocatable form. These modules are "building blocks" that you can combine in various ways to produce a family of compatible operating systems. This section describes the modules in detail, and describes how to combine them in various configurations.

Possible TurboDOS configurations include:

- single-user without spooling
- single-user with spooling
- network server
- simple network user (no local disks)
- complex network user (with local disks)

Numerous subtle variations are possible in each of these categories.

Module Hierarchy

The diagram on page 1-3 illustrates how the functional modules of TurboDOS interact. As the diagram shows, the architecture of TurboDOS can be viewed as a three-level hierarchy.

Process Level

The highest level of the hierarchy is the _process level_. TurboDOS can support many concurrent processes at this level. There is one active process that supports the local user who is executing commands and programs in the local TPA. There are also processes to support users running on other computers and making requests of the local computer over the network. There are processes to handle background printing (de-spooling) on local printers. Finally, there is a process that periodically causes disk buffers to be written out to disk.

Kernel Level

The intermediate level of the hierarchy is the kernel level. The kernel supports the 103 C-functions and T-functions, and controls the sharing of computer resources such as processor time, memory, peripheral devices, and disk files. Processes make requests of the kernel through the entrypoint module OSNTRY, which decodes each C-function and T-function by number and invokes the appropriate kernel module.

Driver Level

The lowest level of the hierarchy is the driver level, and contains all the device-dependent drivers necessary to interface TurboDOS to the particular hardware being used. Drivers must be provided for all peripherals, including console, printers, disks, communications channels, and network interface. A driver is also required for the real-time clock (or other periodic interrupt source).

TurboDOS is designed to interface with almost any kind of peripheral hardware. It operates most efficiently with interrupt-driven, DMA-type interfaces, but can also work fine using polled and programmed-I/O devices.

TurboDOS Loader

The TurboDOS loader OSLOAD.COM is a program containing an abbreviated version of the kernel and drivers. Its purpose is to load the full TurboDOS operating system from a disk file (OSSERVER.SYS) into memory at each system cold-start.

```
 ________________________________________________________________
|                   TurboDOS Module Hierarchy                    |
|                                                                |
|                        Despool  Lcl Usr  Net Svc  Buffers      |
|                        DSPOOL   LCLUSR   NETSVC   FLUSHR       |
|                           |     LCLMSG   NETTBL      |         |
| Process Level            |     LCLTBL      |        |         |
|                          |     CMDINT      |        |         |
|                          |     AUTLOD      |        |         |
|                Loader    |     SGLUSR      |        |         |
|                OSLOAD    |     AUTLOG      |        |         |
|                LDRMSG    |      BIOS       |        |         |
|                   |      |     SUBMIT      |        |         |
|                   |______|________|________|________|         |
|---------------------------------|------------------------------|
|                                 |                              |
|                               Decode                           |
| Kernel Level                 OSNTRY                           |
|      ___________________________|______________________        |
|     |             |             |        |        |        |   |
| Memory        Nonfile         File    Net Req  Clock   Support |
| MEMMGR        NONFIL        FILMGR   NETMGR   RTCMGR  DSPCHR  |
|     |         CPMSUP        FILSUP   NETREQ     |     DSPSGL  |
|     |            |          FILCOM   MSGFMT     |     COMSUB  |
|     |            |          FILLOK   NETTBL     |             |
|     |            |          FFOMGR      |       |             |
|     |            |          DRVLOK      |       |             |
|     |            |             |        |       |             |
|     |      ______|______       |        |       |             |
|     |     |      |      |      |        |       |             |
|     |  Comm Ch Printer Console Record    |       |     Initial |
|     |  COMMGR  LSTMGR  CONMGR  BUFMGR    |       |     SYSNIT  |
|     |     |    LSTTBL  CONTBL  DSKMGR    |       |        |    |
|     |     |    SPOOLR  DOMGR   DSKTBL    |       |        |    |
|     |     |    SPLMSG  INPLN      |      |       |        |    |
|     |     |       |       |       |      |       |        |    |
|-----|-----|-------|-------|-------|------|-------|--------|----|
| Driver Level      |       |       |      |       |        |    |
|     |     |       |       |       |      |       |        |    |
|     |  Comm Ch Printer Console  Disk  Network  Clock   Initial |
|     |  COMDRV  LSTDRA  CONDRA  DSKDRA  CKTDRA  RTCDRV  HDWNIT |
| Memory         LSTDRB    or    DSKDRB  CKTDRB    or           |
| MEMTBL          etc.   CONREM   etc.    etc.   RTCNUL         |
|________________________________________________________________|
```

Process Modules

Module	Function
LCLUSR	Responsible for supporting local user's TPA activities.
LCLMSG	Contains all O/S error messages.
LCLTBL	Local user option table.
CMDINT	Command interpreter, processes commands from local user.
AUTLOD	Autoload routine which processes COLDSTRT.AUT and WARMSTRT.AUT if present.
SGLUSR	Routine to flush/free disk buffers at each console input. Use for single-user configurations instead of FLUSHR.
AUTLOG	Automatic log-on routine. Used when full log-on security is not desired. See AUTUSR patch point.
BIOS	Supports C-function 50 (Direct BIOS Call).
SUBMIT	Routine to emulate CP/M processing of $$$.SUB files.
NETSVC	Services network requests from other processors on the network.
NETTBL	Tables to define local network topology, used by NETSVC+NETREQ.
DSPOOL	Processes background printing.
FLUSHR	Periodically flushes disk buffers. Use for network server configuration instead of SGLUSR.

Kernel Modules

Module	Function
OSNTRY	Kernel entrypoint module which decodes each C-function and T-function by number and invokes the appropriate kernel module.
FILMGR	File manager responsible for requests involving local files.
FILSUP	File support routines used by FILMGR.
FILCOM	Processes common file-oriented requests that are never sent over the network.
FILLOK	File- and record-level interlock routines called by FILMGR.
FFOMGR	FIFO management routines called by FILLOK.
DRVLOK	Drive interlock routines.
BUFMGR	Buffer manager called by FILMGR. Maintains pool of disk buffers used to speed local file access.
DSKMGR	Disk manager responsible for physical access to local disks, called by BUFMGR.
DSKTBL	Table defining drives A-P as local or remote disk drives.
NONFIL	Responsible for functions that are not file-oriented.
CPMSUP	Processes C-functions 7, 8, 24, 28, 29, 31, 37 and 107 which are rarely used. May be omitted.

Kernel Modules (Continued)

Module	Function
CONMGR	Responsible for console I/O.
CONTBL	Links CONMGR to console driver.
DOMGR	Responsible for do-files.
INPLN	Console input line editor used by CMDINT and C-function 10.
LSTMGR	Responsible for printer output.
LSTTBL	Table defining printers A-P and queues A-P as local or remote.
SPOOLR	Print spooler which diverts print output to a spool file when spooling is activated. Also handles direct printing to remote printers.
COMMGR	Responsible for communications channel functions.
NETREQ	Responsible for issuing network request messages for all functions not processed locally.
MSGFMT	Network message format table used by NETREQ.
NETMGR	Network message routing routine used by NETSVC and NETREQ.
RTCMGR	Real-time clock manager responsible for maintaining system date and time.
DSPCHR	Multi-task dispatcher which controls sharing of the local processor among multiple processes.

Kernel Modules (Continued)

Module	Function
DSPSGL	Null dispatcher used as alternative to DSPCHR when only one process is required (OSLOAD.CMD and single-user w/o spooling).
MEMMGR	Memory manager responsible for dynamic allocation of memory, and for supporting TPA allocation C-functions (53-58).
COMSUB	Common subroutines used in all configurations.
SYSNIT	System initialization routine executed at system cold-start.
RTCNUL	Null real-time clock driver, used in configurations where there is no periodic interrupt source.
CONREM	Remote console driver for network server to support SERVER command.
PATCH	128 bytes of zeroes, may be included to provide patch area.

Driver Modules

Module	Function
CONDR_	Console I/O driver.
LSTDR_	Printer output driver(s).
DSKDR_	Disk driver(s).
CKTDR_	Network circuit driver(s).
COMDRV	Communications channel driver.
RTCDRV	Real-time clock driver.
MEMTBL	Table defining the size and structure of main memory (RAM).
HDWNIT	Cold-start initialization for all hardware-dependent drivers.

Standard Packages

To simplify the system generation process, the most commonly-used combinations of TurboDOS modules are pre-packaged into the following standard configurations:

Package	Description
STDLOADR	cold-start loader
STDSINGL	single-user without spooling
STDSPOOL	single-user with spooling
STDMASTR	network server
STDSLAVE	simple user w/o local disks
STDSLAVX	complex user with local disks

The contents of each standard package is detailed in the table on the facing page. Most TurboDOS requirements can be satisfied by linking the appropriate standard package together with a few additional kernel modules plus the requisite driver modules.

Module	K	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
LCLUSR	1.1	-	LCLUSR	LCLUSR	LCLUSR	LCLUSR	LCLUSR
LCLMSG	.3	-	LCLMSG	LCLMSG	LCLMSG	LCLMSG	LCLMSG
LCLTBL	.0	-	LCLTBL	LCLTBL	LCLTBL	LCLTBL	LCLTBL
CMDINT	1.5	-	CMDINT	CMDINT	CMDINT	CMDINT	CMDINT
AUTLOD	.2	-	AUTLOD	AUTLOD	AUTLOD	AUTLOD	AUTLOD
SGLUSR	.1	-	SGLUSR	SGLUSR	-	-	SGLUSR
AUTLOG	.0	-	AUTLOG	AUTLOG	AUTLOG	AUTLOG	AUTLOG
BIOS	.3	-	BIOS	BIOS	BIOS	BIOS	BIOS
NETSVC	1.5	-	-	-	NETSVC	-	-
DSPOOL	1.0	-	-	DSPOOL	DSPOOL	-	DSPOOL
FLUSHR	.2	-	-	-	FLUSHR	-	-
OSLOAD	1.1	OSLOAD	-	-	-	-	-
LDRMSG	.1	LDRMSG	-	-	-	-	-
OSNTRY	.5	OSNTRY	OSNTRY	OSNTRY	OSNTRY	OSNTRY	OSNTRY
FILMGR	2.3	FILMGR	FILMGR	FILMGR	FILMGR	-	FILMGR
FILSUP	2.9	FILSUP	FILSUP	FILSUP	FILSUP	-	FILSUP
FILCOM	.3	FILCOM	FILCOM	FILCOM	FILCOM	FILCOM	FILCOM
FILLOK	1.8	-	-	-	FILLOK	-	-
FFOMGR	1.1	-	-	-	FFOMGR	-	-
DRVLOK	.1	-	-	-	DRVLOK	-	-
BUFMGR	1.1	BUFMGR	BUFMGR	BUFMGR	BUFMGR	-	BUFMGR
DSKMGR	.6	DSKMGR	DSKMGR	DSKMGR	DSKMGR	-	DSKMGR
DSKTBL	.0	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL
NONFIL	.2	NONFIL	NONFIL	NONFIL	NONFIL	NONFIL	NONFIL
CONMGR	.4	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR
CONTBL	.0	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL
PGMLOD	.9	-	PGMLOD	PGMLOD	PGMLOD	PGMLOD	PGMLOD
DOMGR	.3	-	DOMGR	DOMGR	DOMGR	DOMGR	DOMGR
INPLN	.2	-	INPLN	INPLN	INPLN	INPLN	INPLN
LSTMGR	.3	-	LSTMGR	LSTMGR	LSTMGR	LSTMGR	LSTMGR
LSTTBL	.1	-	LSTTBL	LSTTBL	LSTTBL	LSTTBL	LSTTBL
SPOOLR	.6	-	-	SPOOLR	SPOOLR	SPOOLR	SPOOLR
SPLMSG	.1	-	-	SPLMSG	SPLMSG	SPLMSG	SPLMSG
COMMGR	.1	-	COMMGR	COMMGR	COMMGR	COMMGR	COMMGR
NETREQ	1.6	-	-	-	-	NETREQ	NETREQ
MSGFMT	.1	-	-	-	-	MSGFMT	MSGFMT
NETMGR	.6	-	-	-	NETMGR	NETMGR	NETMGR
NETTBL	.0	-	-	-	NETTBL	NETTBL	NETTBL
RTCMGR	.1	-	RTCMGR	RTCMGR	RTCMGR	-	RTCMGR
DSPCHR	.7	-	-	DSPCHR	DSPCHR	DSPCHR	DSPCHR
DSPSGL	.2	DSPSGL	DSPSGL	-	-	-	-
MEMMGR	1.0	-	MEMMGR	MEMMGR	MEMMGR	MEMMGR	MEMMGR
COMSUB	.2	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB
SYSNIT	.1	-	SYSNIT	SYSNIT	SYSNIT	SYSNIT	SYSNIT

Standard Packages (Continued)

To supplement the modules contained in these standard packages, the following kernel modules may have to be added:

Module	Where Required
NETREQ+ MSGFMT	In network servers (MASTR) which must make requests of other processors.
NETSVC	In network users (SLAVE/SLAVX) which must service requests from other processors.
CPMSUP	In all systems which require C-functions 7, 8, 24, 28, 29, 31, 37 and 107 to be supported (SINGL/SPOOL/MASTR/SLAVE/SLAVX).
CONREM	In network servers (MASTR) that have no console device attached, to allow use of SERVER command (in lieu of console driver).
RTCNUL	In all configurations which have no RTC driver (including LOADR).
PATCH	In all configurations which require an additional patch area.

Memory Required

To estimate the memory required by a particular TurboDOS configuration, you need to take into account the combined size of all functional modules, driver modules, disk buffers, and other dynamic storage.

Drivers typically require 1K to 4K, and can be even larger if the hardware is especially complex. Disk buffer space should be as large as possible for optimum performance, especially in a network server. About 4K of disk buffer space is reasonable for a single-user system, although less can be used in a pinch. Other dynamic storage doesn't usually exceed 1K in single-user systems, 2K in network servers.

The following table gives typical memory requirements for standard TurboDOS configurations:

	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
O/S	10K	14K	16K	22K	11K	19K
Drivers	2K	2K	2K	3K	3K	2K
Buffers	4K	4K	4K	16K	-	4K
Dynamic	1K	1K	1K	3K	2K	2K
Total	17K	21K	23K	44K	16K	27K

Other Languages

To facilitate translation into languages other than English, TurboDOS has been implemented with all textual messages segregated into separate modules. All such message modules are available in source form to TurboDOS licensees upon request.

The following modules contain all TurboDOS operating system messages:

Module	Contains
LCLMSG	Most operating system messages.
SPLMSG	Spooler error messages.
LDRMSG	Loader messages for OSLOAD.CMD.

In addition, a separate message module is available for each TurboDOS command.

SYSTEM GENERATION

This section explains the TurboDOS system generation procedure in detail. It describes how to use TLINK to link a desired set of TurboDOS modules together, and details the numerous system patch points which may be modified during system generation. Step-by-step procedures and examples are provided.

Introduction

The functional modules of TurboDOS are distributed in relocatable object form (.O files). Hardware-dependent driver modules are furnished in the same fashion. The TurboDOS TLINK command is a specialized linker used to bind the desired combination of modules together into an executable version of TurboDOS. TLINK also includes a symbolic patch facility used to modify a variety of operating system parameters.

To generate a complete TurboDOS system, you typically must use TLINK several times. At minimum, you have to generate a server operating system OSSERVER.SYS. For a networking system you also have to generate a user operating system OSUSER.SYS. Complex networks may require generation of several different user or server configurations. Finally, you may have to use TLINK to generate a cold-start bootstrap routine for the start-up PROM or boot track.

At cold-start, the bootstrap routine loads the loader program OSLOAD.COM into the TPA of the server computer and executes it. OSLOAD loads the server operating system from the file OSSERVER.SYS into memory. The server operating system then down-loads the user operating system from the file OSSLAVE.SYS over the network into each user computer.

TLINK Command

The TLINK command 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 TurboDOS assembler TASM.

Syntax

```
TLINK inputfn {outputfn} {-options}
```

Explanation

The TLINK command links a specified collection of relocatable object modules together into a single executable file. The "inputfn" argument identifies the two input files used by TLINK: 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}
```

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.

Explanation (Continued)

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".

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 "0dddddd" 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
```

Patch Points

The following table describes 42 public symbols in TurboDOS which you may wish to modify using the symbolic patch facility of TLINK. (Other patch points may exist in hardware-dependent drivers, but they are beyond the scope of this document.)

Symbol	Default Value	Module
ABTCHR	= 0x03 ;CTRL-C	CONTBL
Abort character (after attention).		
ATNBEL	= 0x07 ;CTRL-G	CONTBL
Attention-received warning character.		
ATNCHR	= 0x13 ;CTRL-S	CONTBL
Attention character. May be patched to another character if the default value of CTRL-S is needed by application programs. A common choice is zero (NUL), which allows the console BREAK key to be used as an attention key.		
AUTUSR	= 0xFF	AUTLOG
Automatic log-on user number. Default value of 0xFF requires that user log-on via LOGON command. If automatic log-on desired at cold-start, patch AUTUSR to the desired user number (0-31), and set the sign-bit if a privileged log-on is desired. Generally patched to 0x80 in single-user systems to cause automatic privileged log-on to user zero.		

Patch Points (Continued)

Symbol	Default Value	Module
BFLDLY	= (300)	FLUSHR

Buffer flush delay determines how often disk buffers are written to disk, stated in system "ticks". Default value (300 decimal) causes buffers to be flushed about every five seconds (assuming 60 ticks per second).

Symbol	Default Value	Module
BUFSIZ	= 3	BUFMGR

Default disk buffer size (0=128, 1=256, 2=512, 3=1K,..., 7=16K). Default value specifies 1K disk buffers.

Symbol	Default Value	Module
CKTAST	= (0x0000),(CKTDRA), (0x0100),(CKTDRB), (0x0200),(CKTDRC), (0x0300),(CKTDRD)	NETTBL

Circuit assignment table defines network topology. Contains NMBCKT two-word entries, one for each network circuit to which this processor is attached. The first word of each entry specifies the network address by which this processor is known on a particular circuit, and the second word specifies the entrypoint address of the circuit driver responsible for that circuit. (Possibly several circuits may be handled by the same driver.)

Symbol	Default Value	Module
CLBLEN	= 157	CMDINT

Command line buffer length defines longest permissible command line. The default value permits two 80-char lines.

Patch Points (Continued)

Symbol	Default Value	Module
CLPCHR = "}"		CMDINT
Command line prompt character.		
CLSCHR = "\"		CMDINT
Command line separator character.		
COLDFN = 0,"COLDSTRT","AUT"		AUTLOD
File name and drive for cold-start auto-load processing (in FCB format).		
COMPAT = 0		FILCOM
Default compatibility flags which define rules to be used for file-sharing. Patch to 0xF8 to relax most MP/M restrictions.		
CONAST = 0,(CONDRA)		CONTBL
Console assignment table defines how console I/O is handled. First byte passed to console driver, and commonly defines the channel number (e.g., serial port) to be used for the console. Following word specifies the entrypoint address of the console driver to be used.		
CPMVER = 0x31		NONFIL
CP/M BDOS version number returned by C-function 12 in L-register.		

Patch Points (Continued)

Symbol	Default Value	Module
DEFDID	= (0)	NETTBL

Default network destination ID, used for routing all network requests that are not related to a particular disk drive, queue or printer. In a user, DEFDID should be set to the network address of the server.

Symbol	Default Value	Module
DSKAST	= 0,(DSKDRA),1,(DSKDRB), 0xFF,(0),0xFF,(0),...	DSKTBL

Disk assignment table, an array of 16 three-byte entries (one for each drive letter A-P) that defines which drives are local, remote, and invalid.

For a local drive, the first byte must not have the sign-bit set. That byte is passed to the disk driver, and is commonly used to differentiate between multiple drives connected to a single controller. The following word specifies the entry-point address of the disk driver to be used.

For a remote drive, the first byte must have the sign-bit set. The low-order bits of that byte specify the drive letter to be accessed on the remote processor. The following word specifies the network address of the remote processor.

For an invalid drive, the first byte must be 0xFF, and the following word should be (0).

Patch Points (Continued)

Symbol	Default Value	Module
DSKAST (Continued)		DSKTBL
NOTE: In user configurations STDSLAVE and STDSLAVX, the default values are: DSKAST = 0x80,(0),0x81,(0), 0x82,(0),0x83,(0), ...,0x8E,(0),0x8F,(0)		
DSPPAT = 1,1,1,...,1		LSTTBL
De-spool printer assignment table, an array of 16 bytes (one for each printer letter A-P) that defines the initial queue to which each printer is assigned. Values 1 through 16 correspond to queues A-P, and 0 means that the printer is off-line. The default value assigns all printers to queue A.		
ECOCHR = 0x10 ;CTRL-P		CONTBL
Echo-print character (after attention).		
EOPCHR = 0		LSTTBL
End-of-print character. May be patched to any non-null character, in which case the presence of that character in the print output stream will automatically signal an end-of-print-job condition. The value zero disables this feature.		

12

Patch Points (Continued)

Symbol	Default Value	Module
FWDTBL	= (0xFFFF),(0xFFFF), (0xFFFF),(0xFFFF),0xFF	NETTBL
	Network forwarding table, an array of two-byte entries that define any explicit message forwarding routes to be used by this processor. The first byte of each entry specifies a "foreign" circuit number N, and the second byte a "domestic" circuit number C. Any messages destined for circuit N will be routed via circuit C. This table is variable-length, terminated by 0xFF, and defaults to empty.	
LDCOLD	= 0xFF	AUTLOD
	Cold-start autoload enable flag. Patch to zero if you want to disable the cold-start autoload feature (COLDSTRT.AUT).	
LDWARM	= 0xFF	AUTLOD
	Warm-start autoload enable flag. Patch to zero if you want to disable the warm-start autoload feature (WARMSTRT.AUT).	
LOADFN	= 0,"OSMASTER","SYS"	OSLOAD
	Default file name and drive (in FCB format) loaded by OSLOAD.COM. Drive field (FCB byte 0) may be patched to an explicit drive value to inhibit scanning.	

Patch Points (Continued)

Symbol	Default Value	Module
LOGUSR = 31	User number for logged-off state.	FILCOM
NMBCKT = 1	Number of network circuits to which this processor is connected.	NETTBL
NMBMBS = 0	Number of message buffers pre-allocated at cold-start. Message buffers are allocated dynamically as needed, but this may cause fragmentation which prevents you from changing the size of the disk buffer pool with the BUFFERS command. If this is important, patching NMBMBS to a suitable positive value will eliminate the problem (twice the number of network nodes is a good starting value to try).	NETMGR
NMBRPS = 0	Number of reply packets pre-allocated at cold-start. Reply packets are allocated dynamically as needed, but this may cause fragmentation which prevents you from changing the size of the disk buffer pool with the BUFFERS command. If this is important, patching NMBRPS to a suitable positive value will eliminate the problem (the number of network nodes is a good starting value to try).	NETMGR

Patch Points (Continued)

Symbol	Default Value	Module
NMBSVC = 2		NETSVC
Number of network server processes to be activated. (The number of network nodes is a good starting value to try.)		
NMBUFS = 4		BUFMGR
Default number of disk buffers allocated at cold-start. Must be at least 2. For optimum performance, allocate as many buffers as possible (consistent with TPA and other memory requirements).		
OSMLEN = (128) ;2K bytes		MEMMGR
Length (in paragraphs) of the memory area to be allocated immediately above the TurboDOS operating system resident for disk buffers and other dynamic working storage. The default value (128 paragraphs or 2K bytes) is appropriate for a simple user with no disk buffers. For other configurations, patch OSMLEN to a value large enough to accomodate the disk buffer pool plus at least 2K bytes of miscellaneous dynamic space. Divide the total byte-length of the space required by 16 to give the value of OSMLEN in paragraphs.		
PRTCHR = 0x0C ;CTRL-L		CONTBL
End-print character (after attention). This is a console attention-response, not to be confused with EOPCHR.		

Patch Points (Continued)

Symbol	Default Value	Module
PRTMOD	= 1	LCLTBL
Initial print mode for local user. The default value of 1 specifies spooling. Patch to 0 for direct, or 2 for console.		
PTRAST	= 0,(LSTDRA),0xFF,(0), 0xFF,(0),0xFF,(0),...	LSTTBL
Printer assignment table, an array of 16 three-byte entries (one for each printer letter A-P) that defines which printers are local, remote, and invalid. For a local printer, the first byte must not have the sign-bit set. That byte is passed to the disk printerr, and is commonly defines the channel number (e.g., serial port) to be used for the printer. The following word specifies the entry-point address of the printer driver. For a remote printer, the first byte must have the sign-bit set. The low-order bits of that byte specify the printer letter to be accessed on the remote processor. The following word specifies the network address of the remote processor. For an invalid printer, the first byte must be 0xFF, and the following word should be (0). NOTE: In user configurations STDSLAVE and STDSLAVX, the default values are: PTRAST = 0x80,(0),0x81,(0), 0x82,(0),0x83,(0), ...,0x8E,(0),0x8F,(0)		

Patch Points (Continued)

Symbol	Default Value	Module
QUEAST = 0,(0),0xFF,(0), 0xFF,(0),0xFF,(0),...		LSTTBL

Queue assignment table, an array of 16 three-byte entries (one for each queue letter A-P) that defines which queues are local, remote, and invalid.

For a local queue, all three bytes must be set to zero.

For a remote queue, the first byte must have the sign-bit set. The low-order bits of that byte specify the queue letter to be accessed on the remote processor. The following word specifies the network address of the remote processor.

For an invalid queue, the first byte must be 0xFF, and the following word should be (0).

NOTE: In user configurations STDSLAVE and STDSLAVX, the default values are:

```
QUEAST = 0x80,(0),0x81,(0),
         0x82,(0),0x83,(0),
         ...,0x8E,(0),0x8F,(0)
```

Symbol	Default Value	Module
QUEPTR = 1		LCLTBL

Initial queue or printer assignment. If PRTMOD = 1 (spooling), QUEPTR specifies a queue assignment. If PRTMOD = 0 (direct) QUEPTR specifies a printer assignment. In both cases, values 1 through 16 correspond to letters A-P, and zero means do not queue or print off-line.

Patch Points (Continued)

Symbol	Default Value	Module
RESCHR	= 0x11 ;CTRL-Q	CONTBL
	Resume character (after attention).	
SCANDN	= 0	OSLOAD
	Scan direction flag for OSLOAD. Patch to 0xFF to scan P-to-A (instead of A-to-P).	
SLVFN	= "OSSLAVE ","SYS"	NETSVC
	Name and type of file (in FCB format) to be down-loaded into user processors.	
SPLDRV	= 0xFF	LCLTBL
	Initial spool drive. Default value 0FF indicates spool to system disk (disk from which TurboDOS was loaded at cold-start). Patch to 0 through F to specify a particular drive A-P.	
SRHDRV	= 0	CMDINT
	Search drive for command files. Patch to value 1 through 16 to search drive A-P if command is not found on current (default) drive. Patch to 0xFF to search system disk (disk from which TurboDOS was loaded at cold-start). Default value 0 disables this feature altogether.	

Patch Points (Continued)

Symbol	Default Value	Module
SUBFN	= 0,"$$$ ","SUB"	SUBMIT
Submit file name searched for by optional CP/M submit-file emulator.		
WARMFN	= 0,"WARMSTRT","AUT"	AUTLOD
File name and drive for warm-start auto-load processing (in FCB format).		

Network Operation

TurboDOS accomodates a wide variety of network topologies, ranging from the simplest point-to-point server/user networks to the most complex star, ring, and hierarchical structures.

Network Model

A TurboDOS network is defined to consist of up to 255 circuits, with up to 255 nodes (processors) on each circuit. Each node has a unique 16-bit network address consisting of an 8-bit circuit number plus an 8-bit node number (on that circuit).

Any processor may be connected to several circuits, if desired. A processor connected to multiple circuits has multiple network addresses, one for each circuit. Such a processor even may be set up to perform message forwarding from one circuit to another, permitting dialogue between network nodes that do not share a common circuit between them (more on this later).

Network Tables

The actual network topology is defined by a series of tables in each processor. The tables are set up during system generation, and define the network as "seen" from the viewpoint of each processor. The tables are:

Symbol	Description
NMBCKT	A byte value that defines the number of network circuits to which this processor is connected.

Network Tables
(Continued)

Symbol	Description
CKTAST	The circuit assignment table containing NMBCKT entries defining the network address by which this processor is known on each circuit, and specifying the network circuit driver responsible for each handling each circuit.
DSKAST	The disk assignment table that specifies for all drive letters A-P which are local, remote, and invalid. This table specifies a network address for each remote drive, and a disk driver for each local drive.
PTRAST	The printer assignment table that specifies for all printer letters A-P which are local, remote, and invalid. This table specifies a network address for each remote printer, and a printer driver for each local printer.
QUEAST	The queue assignment table that specifies for all queue letters A-P which are local, remote, and invalid. This table specifies a network address for each remote queue.
DEFDID	The default network destination ID, used for routing all network requests that are not related to a specific disk drive, printer, or queue.

Network Tables
(Continued)

Symbol	Description
FWDTBL	The message forwarding table that specifies any additional circuits (not directly connected to this processor) which may be accessed via explicit message forwarding, and how messages destined for such circuits are to be routed.

These tables are pre-defined with default values to make set-up of simple server/user networks very easy. For complex multi-circuit networks, the set-up is somewhat more complicated (as might be expected).

Refer to the preceding Patch Points subsection for details of the organization and defaults for these network tables.

Message Forwarding

The network architecture of TurboDOS supports two kinds of message forwarding: "implicit" and "explicit". To understand the distinction, consider the case of a network with three processors (P1, P2, and P3) connected by two circuits (C1 and C2) as follows:

```
 ____          ____          ____
|    |        |    |        |    |
| P1 |---C1---| P2 |---C2---| P3 |
|____|        |____|        |____|
```

A program running in P1 makes an access to drive D. Suppose the disk assignment tables in the three processors are set up in the following fashion:

- P1's DSKAST defines its drive D as a remote reference to P2's drive B.
- P2's DSKAST defines its drive B as a remote reference to P3's drive A.
- P3's DSKAST defines its drive A as a local device attached directly to P3.

In this case, P1's access to its drive D actually winds up implicitly accessing P3's drive A. This is *implicit* forwarding.

Alternatively, suppose P1's DSKAST defines its drive D as a remote reference to P3's drive A, and that P1's FWDTBL provides that messages destined for circuit C2 may be routed via C1. In this case, P1 sends a request to P3 on circuit C1. P2 receives the request, recognizes that it should be forwarded, and retransmits the request to P3 via circuit C2. Thus, P1 accesses P3's drive A with the assistance of P2, but this time P1 is not aware of P2's role in the transaction. This is *explicit* forwarding.

DISTRIBUTION

This section explains the TurboDOS distribution procedure in detail. It covers TurboDOS licensing requirements, and the obligations of licensed distributors, dealers, and end-users. It describes how to make up and serialize TurboDOS distribution disks.

Although this section is of concern primarily to licensed TurboDOS distributors, we've included it here so that dealers and end-users can gain a better perspective on the overall distribution process.

TurboDOS Licensing

TurboDOS is a proprietary software product of Software 2000, Inc. As such, it is protected by law against unauthorized use and reproduction. Authorization to use and/or reproduce TurboDOS is granted only by written license agreement.

Legal Protection

TurboDOS programs and documentation are copyrighted, which means it is against the law to make copies without express written authorization from Software 2000 to do so.

The word "TurboDOS" is a trademark owned by Software 2000 and registered in Class 9 (computer software) and Class 16 (documentation) with the trademark offices of the United States and most of the developed countries of the free world. This means it is against the law to make use of the TurboDOS trademark without express written authorization from Software 2000.

Software 2000 has licensed certain companies to distribute TurboDOS. Such distributors are authorized to use the TurboDOS trademark, and to reproduce, distribute, and sub-license TurboDOS programs and documentation to dealers and end-users.

User Obligations

TurboDOS may be used only after the user has paid the required license fee, signed a copy of the TurboDOS end-user license agreement, and returned the signed agreement to the issuing TurboDOS distributor. Then, TurboDOS may be used only in strict conformance with the terms of the license.

Each end-user license allows TurboDOS to be used on one specific computer system identified by make, model, and serial number. The end-user license may not be transferred from one computer system to another, and expressly forbids copying programs and documentation except as required for backup purposes only.

A separate license fee must be paid and a separate license signed for each computer system on which TurboDOS is used. Network slave computers that cannot operate stand-alone (because, for example, they have no local disk) do not have to be licensed separately from the network server. However, networked computers that are also capable of stand-alone operation under TurboDOS must each be licensed separately (whether or not they are actually used stand-alone).

Dealer Obligations

A dealer must sign a TurboDOS dealer agreement and return the signed agreement to the issuing distributor. Then, the dealer is permitted to purchase pre-serialized copies of TurboDOS programs and documentation from the distributor, and to resell them to end-users. Dealers may not make copies of TurboDOS programs or documentation for any purpose whatever.

Before delivering each copy of TurboDOS, the dealer must see to it that the end-user signs the TurboDOS end-user license agreement and returns it to the issuing distributor.

Distributor Obligations

Each licensed TurboDOS distributor is provided a master copy of TurboDOS relocatable modules and command programs on diskette. A distributor is allowed to reproduce and distribute copies of TurboDOS to dealers and end-users, but only in connection with certain specifically authorized hardware (usually manufactured or sold by the distributor). The distributor is required to serialize each copy of TurboDOS with a unique sequential magnetic serial number, and to register each serial number promptly with Software 2000. (Serialization is described in more detail below.)

Each distributor is also provided with a master copy of TurboDOS documentation, either in camera-ready hardcopy or in ASCII files on disk. The distributor is responsible for reproducing the documentation and furnishing it with each copy of TurboDOS it issues.

A distributor must require each dealer to sign and return a TurboDOS dealer agreement before issuing copies of TurboDOS to the dealer for resale. A distributor must require each end-user to sign and return a TurboDOS end-user license agreement before issuing a copy of TurboDOS directly to the end-user.

Serialization

Each copy of TurboDOS is magnetically serialized with a unique serial number. Such serialization helps ensure that reproduction and distribution of TurboDOS is done in strict accordance with the required licensing and registration procedures, and facilitates tracing of unlicensed copies of the software.

Each relocatable module of TurboDOS distributed to a dealer or end-user has a magnetic serial number composed of two parts:

- an _origin number_ that identifies the issuing distributor, and
- a sequential _unit number_ that uniquely identifies each copy of TurboDOS issued by that distributor.

During system generation, the TLINK command verifies that all modules making up a TurboDOS configuration are serialized consistently, and magnetically serializes the resulting executable version of TurboDOS accordingly.

The relocatable modules on the master disk furnished to each licensed TurboDOS distributor are partially serialized with an origin number only. Each distributor is provided a serialization program (SERIAL.CMD) that must be used to add a unique sequential unit number to each copy of TurboDOS issued by the distributor. The TLINK command will not accept partially-serialized modules that have not been serialized with a unit number. Conversely, the SERIAL command will not reserialize modules that have already been fully serialized.

Technical Support

Software 2000 maintains telephone and telex "hot-lines" to provide TurboDOS technical assistance to its distributors. These are unlisted numbers providing direct access to the authors of the TurboDOS operating system, and are furnished only to licensed TurboDOS distributors. We encourage distributors to take advantage of this service whenever technical questions or problems arise in using or configuring TurboDOS.

It is the responsibility of each licensed distributor to provide technical support to its dealers and end-user customers. Software 2000 cannot assist dealers or end-users directly. Where exceptional circumstances seem to require direct contact between Software 2000 technical personnel and a dealer or end-user, this must be handled strictly by prior arrangement between Software 2000 and the distributor.

SERIAL Command

The SERIAL command enables TurboDOS distributors to magnetically serialize relocatable modules of TurboDOS for distribution.

Syntax

```
SERIAL srcefile destfile ;Unnn {options}
SERIAL ;Unnn {options}
```

Explanation

The SERIAL command works exactly like the COPY command, and accepts exactly the same arguments and options. However, SERIAL has the additional function of magnetically serializing relocatable modules as they are copied. SERIAL serializes files of type .REL (Z80 modules) and type .O (8086 modules). Other files are copied without any change.

The unit number must be specified on the command line as ;Unnn, where "nnn" represents a decimal unit number in the range 0-65535. Unit numbers must be assigned sequentially, starting with 1. Unit number 0 is reserved by convention for in-house use by the distributor.

SERIAL produces fully-serialized modules that are encoded with the distributor's origin number and the specified unit number. TLINK does not accept TurboDOS modules unless they have been fully serialized in this fashion.

Options

Option	Explanation
SERIAL accepts all COPY options, plus:	
;Unnn	Relocatable modules (type .REL or .O) are magnetically serialized with unit number nnn, which must be a decimal integer in the range 0 to 65535. This "option" is mandatory for SERIAL.

Example

```
0A}SERIAL *.O B: ;U289N
0A:AUTLOD  .O   copied to 0B:AUTLOD  .O
0A:AUTLOG  .O   copied to 0B:AUTLOG  .O
                     :
0A:SYSNIT  .O   copied to 0B:SYSNIT  .O
0A}
```

Error Messages

SERIAL incorporates all COPY error messages, plus:

Unit number not specified
Origin number violation
File is already serialized
Unexpected EOF in .O or .REL file

PACKAGE Command

The PACKAGE command lets you combine any collection of relocatable object modules into a single concatenated .O file.

Syntax

```
PACKAGE srcefile {destfile}
```

Explanation

PACKAGE may be used to construct custom packages of TurboDOS modules, make additions or changes to the supplied STDxxxxx packages, pre-package collections of driver modules, and so forth.

The "srcefile" argument specifies the name of an input file "srcefile.PKG" that lists the modules to be packaged. The "destfile" argument specifies the name of the concatenated .O file to be created. If "destfile" is omitted, then the "srcefile" argument is also used as the name of the output .O file.

If the .PKG file is found, it must contain the list of relocatable object modules (.O files) to be linked together. If the .PKG file is not found, then the PACKAGE command 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 is:

```
objectfn {,objectfn}... {;comment}
```

A null directive terminates the prompting sequence and causes processing to proceed.

After obtaining the list of modules from the file or console, PACKAGE concatenates all of the modules together (displaying the name of each module as it is encountered) and writes the result to the output file.

Example

```
0A}PACKAGE STDLOADR
* ; STDLOADR.PKG standard loader package
* OSLOAD,LDRMSG,OSNTRY,FILMGR,FILSUP
* FILCOM,BUFMGR,DSKMGR,DSKTBL,NONFIL
* CONMGR,CONTBL,DSPSGL,COMSUB
OSLOAD LDRMSG OSNTRY FILMGR FILSUP etc.
0A}
```

Error Messages

```
File name missing from command
Invalid input file name
Unexpected EOF in input file
Disk is full
Can't make output file
Can't open input file
No input files
```

Distribution Procedure

Here is the procedure to be followed by distributors when creating each copy of TurboDOS to be issued to a dealer or end-user:

1. Assign a unique sequential unit number for this copy of TurboDOS, and register it immediately by filling out a serial number registration card (or agreed-to substitute) and mailing to Software 2000, Inc.

2. Format a new disk, and label it with the following information clearly legible:

   - trademark TurboDOS™

   - version number (1.3x)

   - origin and unit numbers (oo/uuuu)

   - statutory copyright notice:
     Copyright 198x by Software 2000, Inc.
     All rights reserved.

3. Use the SERIAL command to copy and serialize the appropriate files from your distribution master disk to the new disk. Use the tables on the following page to guide you in determining what files to put on the new disk.

   IMPORTANT NOTE: Be absolutely certain that the new disk does <u>not</u> contain any unserialized modules or SERIAL.CMD!

4. Using the new serialized disk, use the TLINK command to generate an executable loader and operating system. Follow the system generation procedure described in the previous section.

5. In addition to the serialized disk, you should issue copies of TurboDOS documentation and a start-up PROM (if applicable).

Distribution Procedure (Continued)

The following table may be used for guidance in preparing TurboDOS disks for distribution. In addition to the files shown, you need to include hardware-dependent driver modules and utility programs as appropriate.

single-user w/o spooler	single-user with spooler	multi-user networking
STDLOADR.O	STDLOADR.O	STDLOADR.O
STDSINGL.O	STDSINGL.O	STDSINGL.O
-	STDSPOOL.O	STDSPOOL.O
-	-	STDMASTR.O
-	-	STDSLAVE.O
-	-	STDSLAVX.O
CPMSUP .O	CPMSUP .O	CPMSUP .O
RTCNUL .O	RTCNUL .O	RTCNUL .O
PATCH .O	PATCH .O	PATCH .O
SUBMIT .O	SUBMIT .O	SUBMIT .O
OSBOOT .O	OSBOOT .O	OSBOOT .O
-	-	NETREQ .O
-	-	MSGFMT .O
-	-	NETSVC .O
-	-	CONREM .O
AUTOLOAD.CMD	AUTOLOAD.CMD	AUTOLOAD.CMD
BACKUP .CMD	BACKUP .CMD	BACKUP .CMD
-	-	BATCH .CMD
BOOT .CMD	BOOT .CMD	BOOT .CMD
BUFFERS .CMD	BUFFERS .CMD	BUFFERS .CMD
-	-	CHANGE .CMD
COPY .CMD	COPY .CMD	COPY .CMD
DATE .CMD	DATE .CMD	DATE .CMD
DELETE .CMD	DELETE .CMD	DELETE .CMD
DIR .CMD	DIR .CMD	DIR .CMD
DO .CMD	DO .CMD	DO .CMD
DRIVE .CMD	DRIVE .CMD	DRIVE .CMD
DUMP .CMD	DUMP .CMD	DUMP .CMD
ERASEDIR.CMD	ERASEDIR.CMD	ERASEDIR.CMD
-	-	FIFO .CMD
FIXDIR .CMD	FIXDIR .CMD	FIXDIR .CMD

Distribution Procedure (Continued)

single-user w/o spooler		single-user with spooler		multi-user networking	
FIXMAP	.CMD	FIXMAP	.CMD	FIXMAP	.CMD
FORMAT	.CMD	FORMAT	.CMD	FORMAT	.CMD
LABEL	.CMD	LABEL	.CMD	LABEL	.CMD
-		-		LOGOFF	.CMD
-		-		LOGON	.CMD
-		-		SERVER	.CMD
OTOASM	.CMD	OTOASM	.CMD	OTOASM	.CMD
PRINT	.CMD	PRINT	.CMD	PRINT	.CMD
-		PRINTER	.CMD	PRINTER	.CMD
-		QUEUE	.CMD	QUEUE	.CMD
READPC	.CMD	READPC	.CMD	READPC	.CMD
-		-		RECEIVE	.CMD
RENAME	.CMD	RENAME	.CMD	RENAME	.CMD
-		-		SEND	.CMD
SET	.CMD	SET	.CMD	SET	.CMD
SHOW	.CMD	SHOW	.CMD	SHOW	.CMD
TASM	.CMD	TASM	.CMD	TASM	.CMD
TBUG	.CMD	TBUG	.CMD	TBUG	.CMD
TLINK	.CMD	TLINK	.CMD	TLINK	.CMD
TPC	.CMD	TPC	.CMD	TPC	.CMD
TYPE	.CMD	TYPE	.CMD	TYPE	.CMD
USER	.CMD	USER	.CMD	USER	.CMD
VERIFY	.CMD	VERIFY	.CMD	VERIFY	.CMD

CODING CONVENTIONS

This section is devoted to in-depth discussion of TurboDOS internal coding conventions, aimed at the systems programmer writing hardware-dependent drivers or resident processes. All coding examples and driver listings in this document make use of the TurboDOS 8086-family assembler TASM.

Undefined External References

To allow various TurboDOS modules to be included or omitted at will, TLINK automatically resolves all undefined external references to the default names "UndCode" (for code references) and "UndData" (for data references). The common subroutine module COMSUB contains the following:

```
        LOC     Data#       ;data segment
UndData::                   ;undefined data
        WORD    0,0

        LOC     Code#       ;code segment
UndCode::                   ;undefined code
        XOR     AL,AL       ;zero AL & flags
        RET                 ;return
```

Thus, it is always safe to load or call an external name, whether or not it is present at TLINK time. It is bad form to store into an undefined external name, however!

Memory Allocation

A common memory management module MEMMGR provides dynamic allocation and deallocation of memory space required for disk and message buffers, print queues, file and record locks, do-file nesting, and so forth. TurboDOS reserves a region of memory for such dynamic workspace, located immediately above the TurboDOS resident. The length of this area (in paragraphs) is determined by the patchable parameter OSMLEN. Memory segments are allocated downward from the top of the reserved region. Deallocated segments are concatenated with any neighbors and threaded on a free-memory list. A best-fit algorithm is used to reduce memory fragmentation.

Allocation and deallocation requests are coded in this manner:

```
;code to allocate a memory segment
        MOV     BX,=36   ;BX=segment size
        CALL    ALLOC#   ;allocate segment
        TEST    AL,AL    ;alloc successful?
        JNZ     ERROR    ;NZ -> not enuf mem
        PUSH    BX       ;else, BX=&segment
          :
;code to deallocate a memory segment
        POP     BX       ;BX=&segment
        CALL    DEALOC#  ;deallocate segment
```

ALLOC# prefixes each allocated segment with a word containing the segment length, so that DEALOC# can tell how much memory is to be deallocated. ALLOC# does not zero the newly-allocated segment.

List Processing

TurboDOS maintains its dynamic structures as threaded lists with bidirectional linkages. This technique permits a node to be added or deleted anywhere in a list without searching. The list head and each list node have a two-word linkage (forward and backward pointers).

List manipulation is coded in this manner:

```
        LOC   Data#     ;data segment
;list head (linkage initialized empty)
LSTHED: WORD  LSTHED    ;forward pointer
        WORD  LSTHED    ;backward pointer

;list node (linkage not initialized)
LSTNOD: WORD  0         ;forward pointer
        WORD  0         ;backward pointer
        RES   128       ;contents of node

        LOC   Code#     ;program segment
;code to add node to end of list
        MOV   BX,&LSTHED  ;BX=&head
        MOV   DX,&LSTNOD  ;DX=&node
        CALL  LNKEND#   ;link to list end

;code to unlink node from list
        MOV   BX,&LSTNOD ;BX=&node
        CALL  UNLINK#   ;unlink node

;code to add node to beginning of list
        MOV   BX,&LSTHED ;BX=&head
        MOV   DX,&LSTNOD ;DX=&node
        CALL  LNKBEG#   ;link to list beg.
```

Task Dispatching

TurboDOS incorporates a flexible, efficient mechanism for dispatching the 8086-family CPU among various competing processes. In coding drivers for TurboDOS, you must take extreme care to use the dispatcher correctly in order to attain maximum system performance.

The dispatcher allows one process to wait for some event (for example, data-available or seek-complete) while allowing other processes to use the processor. For each such event, you must define a three-word structure called a "semaphore".

A semaphore consists of a count-word followed by a two-word list head. The count-word is used by the dispatcher to keep track of the status of the event, while the list head anchors a threaded list of processes waiting for the event to occur.

Two primitive operations operate on a semaphore: waiting for the event to occur (WAIT#), and signalling that the event has occurred (SIGNAL#). They are coded in this following manner:

```
;this semaphore represents some event
EVENT:  WORD   0          ;semaphore count
        WORD   EVENT+2    ;semaphore f-ptr
        WORD   EVENT+2    ;semaphore b-ptr

;wait for the event to occur
        MOV    BX,&EVENT  ;BX=&semaphore
        CALL   WAIT#      ;wait for event

;signal that event has occurred
        MOV    BX,&EVENT  ;BX=&sempahore
        CALL   SIGNAL#    ;signal event
```

Task Dispatching (Continued)

Whenever a process waits on a semaphore, WAIT# decrements the semaphore's count-word. Thus, a negative count -N signifies that there are N processes waiting for the event to occur. Whenever an event is signalled, SIGNAL# increments the semaphore count-word and awakens the process that has been waiting longest.

If an event is signalled but no process is waiting for it, then SIGNAL# increments the count-word to a positive value. Thus, a positive count N signifies that there have been N occurrences of the event for which no process was waiting. In this case, the next N calls to WAIT# on that semaphore will return immediately without waiting.

Sometimes it is necessary for a process to wait for a specific time interval (for example, a motor-start delay or carriage-return delay) rather than for a specific event. TurboDOS provides a delay facility (DELAY#) that permits other processes to use the CPU while one process is waiting for such a timed delay. Delay intervals are specified as some number of "ticks". A tick is an implementation-defined interval, usually 1/50 or 1/60 of a second. Delays are coded thus:

```
;delay for one-tenth of a second
        MOV   BX,=6     ;BX=delay in ticks
        CALL  DELAY#    ;delay process
```

Accuracy of delays is usually plus-or-minus one tick. A delay of zero ticks may be specified to relinquish the processor to other processes on a "courtesy" basis.

All driver delays should be accomplished via WAIT# or DELAY#, <u>never</u> by spinning in a loop.

Interrupt Service

Dispatching is especially efficient when used with interrupt-driven devices. Usually, the interrupt service routine just calls SIGNAL# to signal the interrupt-associated event.

Most interrupt service routines should exit via the usual IRET instruction. However, some periodic interrupt (usually a 50 or 60 hertz clock interrupt) should have an interrupt service routine that exits by jumping to the dispatcher entrypoint ISRXIT# to provide periodic time-slicing of processes. To avoid excessive dispatcher overhead, don't use ISRXIT# more than about 60 times per second.

Before calling any TurboDOS support routine (such as SIGNAL#) or referencing any DS-relative data, an interrupt service routine must call the subroutine GETSDS# to set up register DS.

A simple interrupt service routine might be coded like this:

```
DEVISR: PUSH   AX         ;save registers
        PUSH   BX         ;  "      "
        PUSH   CX         ;  "      "
        PUSH   DX         ;  "      "
        PUSH   DS         ;  "      "
        CALL   GETSDS#    ;get system DS
        MOV    BX,&EVENT  ;BX=&semaphore
        CALL   SIGNAL#    ;signal event
        MOV    DX,&EOIR   ;DX=&end-of-int
        MOV    AX,=INTN   ;AX=interrupt#
        OUT    DX,AX      ;reset interrupt
        POP    DS         ;restore registers
        POP    DX         ;   "        "
        POP    CX         ;   "        "
        POP    BX         ;   "        "
        POP    AX         ;   "        "
        IRET              ;return from int.
```

Poll Routines

Devices incapable of interrupting the CPU have to be polled by the driver. The dispatcher maintains a threaded list of poll routines, and executes them every dispatch. The function of each poll routine is to check the status of its device, and to signal the occurrence of some event (for example, data-available) when it occurs. The routine LNKPOL# links a poll routine onto the poll list, and UNLINK# removes it.

A poll routine must be coded so that it will not signal the occurrence of a particular event more than once. The best way to assure this is for the poll routine to unlink itself from the poll list as soon as it has signalled the event. An example:

```
EVENT:  WORD   0          ;semaphore
        WORD   EVENT+2
        WORD   EVENT+2

;driver waits for event
        MOV    DX,&POLNOD ;DX=&poll node
        CALL   LNKPOL#    ;activate poll rtn
        CALL   POLRTN     ;optional pretest
        MOV    BX,&EVENT  ;BX=&semaphore
        CALL   WAIT#      ;wait for event
          :

;poll routine signals event when detected
POLNOD: WORD   0          ;poll rtn linkage
        WORD   0          ;  "    "    "
POLRTN: IN     AL,=STAT   ;AL=device status
        TEST   AL,=MASK   ;did event occur?
        JZ     __X        ;if not, exit
        MOV    BX,&EVENT  ;BX=&semaphore
        CALL   SIGNAL#    ;signal event
        MOV    BX,&POLNOD ;BX=&poll node
        CALL   UNLINK#    ;unlink poll rtn
__X:    RET               ;all done
```

Mutual Exclusion

TurboDOS is fully re-entrant at the process and kernel levels. However, most driver modules are not coded re-entrantly (since most peripheral devices can only do one thing at a time). Consequently, most drivers must make use of a mutual-exclusion interlock to prevent TurboDOS from invoking them re-entrantly.

This is very easy to accomplish using the basic semaphore mechanism of the dispatcher. It is only necessary to define a semaphore with its count-word initialized to 1 (instead of 0). Mutual exclusion may then be accomplished by calling WAIT# upon entry and SIGNAL# upon exit. An example:

```
;mutual-exclusion semaphore
MXSPH:  WORD  1          ;count-word=1!
        WORD  MXSPH+2
        WORD  MXSPH+2

DRIVER: MOV   BX,&MXSPH  ;BX=&semaphore
        CALL  WAIT#      ;wait if in-use
          :

          :
        MOV   BX,&MXSPH  ;BX=&semaphore
        CALL  SIGNAL#    ;unlock mut-excl
        RET              ;done
```

Sample Driver Using Interrupts

Here is a simple device driver for an interrupt-driven serial input device. It illustrates coding techniques discussed so far:

```
MXSPH:  WORD  1           ;MX semaphore
        WORD  MXSPH+2
        WORD  MXSPH+2
RDASPH: WORD  0           ;RDA semaphore
        WORD  RDASPH+2
        WORD  RDASPH+2
CHRSAV: BYTE  0           ;saved input char

;device driver main code
INPDRV::MOV   BX,&MXSPH   ;BX=&MXsemaphore
        CALL  WAIT#       ;lock MX
        STI               ;need ints enabled
        MOV   BX,&RDASPH ;BX=&semaphore
        CALL  WAIT#       ;wait data avail
        PUSH  CHRSAV      ;stack input char
        MOV   BX,&MXSPH   ;BX=&MXsemaphore
        CALL  SIGNAL#     ;unlock MX
        POP   AX          ;return AL=char
        RET               ;done

;interrupt service routine
INPISR::PUSH  AX          ;save registers
        PUSH  BX          ;  "       "
        PUSH  CX          ;  "       "
        PUSH  DX          ;  "       "
        PUSH  DS          ;  "       "
        CALL  GETSDS#     ;get system DS
        IN    AL,=INPUT  ;get input char
        MOV   CHRSAV,AL  ;save for driver
        MOV   BX,&RDASPH  ;BX=&semaphore
        CALL  SIGNAL#     ;signal data avail
        POP   DS          ;restore registers
        POP   DX          ;   "         "
        POP   CX          ;   "         "
        POP   BX          ;   "         "
        POP   AX          ;   "         "
        IRET              ;return from int.
```

Sample Driver Using Polling

Here is a simple device driver for non-interrupting serial input device. It illustrates how polling is used:

```
MXSPH:  WORD   1          ;MX semaphore
        WORD   MXSPH+2
        WORD   MXSPH+2
RDASPH: WORD   0          ;RDA semaphore
        WORD   RDASPH+2
        WORD   RDASPH+2
CHRSAV: BYTE   0          ;saved input char

;device driver main code
INPDRV::MOV    BX,&MXSPH  ;BX=&MXsemaphore
        CALL   WAIT#      ;lock MX
        MOV    DX,&POLNOD ;DX=&pollnode
        CALL   LNKPOL#    ;activate poll rtn
        CALL   POLRTN     ;optional pretest
        MOV    BX,&RDASPH ;BX=&semaphore
        CALL   WAIT#      ;wait data avail
        PUSH   CHRSAV     ;stack input char
        MOV    BX,&MXSPH  ;BX=&MXsemaph
        CALL   SIGNAL#    ;unlock MX
        POP    AX         ;return AL=char
        RET               ;done

;device poll routine with linkage
POLNOD: WORD   0          ;poll rtn linkage
        WORD   0
POLRTN: IN     AL,=STAT   ;get device status
        TEST   AL,=MASK   ;data available?
        JZ     __X        ;if not, exit
        IN     AL,=DATA   ;get input char
        MOV    CHRSAV,AL  ;save for driver
        MOV    BX,&RDASPH ;BX=&semaphore
        CALL   SIGNAL#    ;signal data avail
        MOV    BX,&POLNOD ;BX=&pollnode
        CALL   UNLINK#    ;unlink poll rtn
__X:    RET               ;done
```

Inter-Process Messages

To pass messages from one process to another, a five-word structure called a "message node" is used. A message node consists of a three-word semaphore followed by a two-word message list head. Routines are provided for sending messages to a message node (SNDMSG#), and receiving messages from a message node (RCVMSG#). Typically, the sending process allocates a memory segment in which to build the message, and the receiving process deallocates the segment after reading the message. The first two words of each message must be reserved for a list-processing linkage. Coding is done in this manner:

```
;message node
MSGNOD: WORD   0          ;semaphore part
        WORD   MSGNOD+2   ;    "       "
        WORD   MSGNOD+2   ;    "       "
        WORD   MSGNOD+6   ;message list head
        WORD   MSGNOD+6   ;   "      "    "

;one process allocates/builds/sends msg
        MOV    BX,=12+4   ;BX=message size+4
        CALL   ALLOC#     ;allocate segment
        PUSH   BX         ;save &segment
          :               ;build msg in seg
        POP    DX         ;DX=&segment
        MOV    BX,&MSGNOD ;BX=&msgnode
        CALL   SNDMSG#    ;send message

;other process reads/deallocates message
        MOV    BX,&MSGNOD ;BX=&msgnode
        CALL   RCVMSG#    ;receive message
        PUSH   BX         ;save &segment
          :               ;process message
        POP    BX         ;BX=&segment
        CALL   DEALOC#    ;deallocate seg
```

Console Routines

TurboDOS includes several handy console I/O subroutines which may be called from within driver modules as illustrated:

```
;raw console I/O routines
        CALL   CONST#    ;get status in AL
        TEST   AL,AL     ;input char avail?
        JZ     __X       ;if not, exit
        CALL   CONIN#    ;get input in AL
        CALL   UPRCAS#   ;make upper-case
        MOV    CL,AL     ;char to CL
        CALL   CONOUT#   ;output char in CL

;message output routines
;message must be null-terminated
        CALL   DMS#      ;output following
MSG:    BYTE   "This is a test message\0"
        MOV    BX,&MSG   ;BX=&message
        CALL   DMSBX#    ;output msg *BX

;binary-to-decimal output routine
        MOV    BX,=31416  ;BX=word value
        CALL   DECOUT#  ;displays decimal
```

Sign-On Message

You may add your own custom sign-on message to TurboDOS. Your message will be displayed at cold-start immediately following the normal TurboDOS sign-on and copyright notice.

Your sign-on message must be coded as an ASCII character string terminated with a $ delimiter, and labelled with the public entry symbol USRSOM. An example:

```
USRSOM::BYTE    0x0D, 0x0A
        BYTE    "Implementation by "
        BYTE    "Trigon Computer Corp."
        BYTE    "$"
```

Resident Process

You can code a resident process that runs in the background concurrent with other system activities, and link it into TurboDOS. The create-process subroutine CRPROC# may be called to create such a process at cold-start as shown:

```
HDWNIT::MOV     BX,=128    ;BX=workspace size
        CALL    ALLOC#     ;alloc workspace
                           ;BX=&workspace
        MOV     DX,&MYPROC ;DX=&entrypoint
        CALL    CRPROC#    ;create process
          :

MYPROC: INC     COUNT[DI]  ;increment count
        MOV     DX,=60*60  ;ticks/minute
        MOV     CL,=2      ;T-function 2
        CALL    OTNTRY#    ;delay 1 minute
        JMP     MYPROC     ;loop forever
```

CRPROC# automatically allocates a TurboDOS process area (address appears in register SI) and a stack area (address appears in SP). If the process requires a re-entrant workspace, it should be allocated with ALLOC# and passed to CRPROC# in BX (as shown above), and will appear to the new process in register DI.

The resident process must make all operating system requests by calling OCNTRY# or OTNTRY# with a C-function or T-function number in register CL. It <u>must not</u> execute INT 0xE0 or INT 0xDF, nor make direct calls on kernel routines such as WAIT#, SIGNAL#, DELAY#, SNDMSG#, RCVMSG#, ALLOC#, and DEALOC#.

Resident Process (Continued)

A resident process is not attached to a console, so any console I/O requests will be ignored.

You can do file processing within a resident process, using the normal C-functions open, close, read, write, and so forth, called via OCNTRY#. First, however, you must remember to warm-start with C-function 0 (OCNTRY#), and then log-on with T-function 14 (OTNTRY#).

A resident process must always be coded to preserve the contents of index register SI, which Turbodos relies upon as a pointer to its process area. The process may use all other registers as desired.

User-Defined Function

The User-Defined Function (T-function 41) provides a means of adding your own special functions to the normal TurboDOS repertoire of C-functions and T-functions. To do this, you simply create a function processor subroutine with the public entrypoint symbol USRFCN.

Whenever a program invokes T-function 41, TurboDOS transfers control to your USRFCN routine. On entry, register CX contains the address of the 128-byte record area passed from the caller's current DMA address, and registers BX and DX contain whatever values the caller loaded into them. Your USRFCN routine may return data to the caller in the 128-byte record area (address in CX at entry) and in any of the registers AX-BX-CX-DX.

Architecturally, your USRFCN routine is inside the TurboDOS kernel. Consequently, it may call kernel subroutines directly. Any calls to C-functions and T-functions must therefore be made by means of two special recursive entrypoints: XCNTRY# and XTNTRY#.

DRIVER INTERFACE

This section explains how to code hardware-dependent device driver modules, and presents formal interface specifications for each category of driver required by TurboDOS.

Following this section is a large appendix that contains assembler source listings of actual driver modules. The sample drivers cover a wide range of peripheral devices, and provide an excellent starting point for your driver development work.

General Notes

Drivers modules are coded with standard public entrypoint names, and linked to TurboDOS using the TLINK command. You may package your drivers into as many or few separate modules as you like. In general, it is easier to reconfigure TurboDOS for a variety of devices if the driver for each device is packaged as a separate module.

TurboDOS is designed to accomodate multiple disk, console, printer, and network drivers. For disk drivers, for instance, the DSKAST is normally set up to refer to disk driver entrypoints DSKDRA#, DSKDRB#, DSKDRC#, and so forth. Each disk driver should be coded with the public entrypoint DSKDR_. TLINK automatically maps successive definitions of such names by replacing the trailing _ by A, B, C, etc. The same technique may be used for console, printer, and network driver entrypoints.

You must code driver routines to preserve CS, DS, SS, SP, SI and DI registers, but you may use other registers as desired.

Initialization

Hardware initialization and interrupt vector set-up should be performed in an initialization routine labelled with the public entry symbol HDWNIT::. TurboDOS calls this routine during cold-start with interrupts disabled.

Your HDWNIT:: routine _must_ _not_ enable interrupts or make calls to WAIT# or DELAY#. In most cases, HDWNIT:: will contain a series of calls to individual driver initialization subroutines contained in other modules.

Memory Table

All 8086 TurboDOS systems must include a table that specifies the size and layout of main memory. The table must be labelled with the public symbol MEMTBL. It must begin with a byte value that specifies the number of discontiguous regions of main memory (up to eight), followed by two words for each region which specify the base address and length of the segment (both in paragraphs). The first segment in the table must be large enough to contain the resident portion of 8086 TurboDOS plus the dynamic workspace (given by OSMLEN).

The following example illustrates the simple case of a system with 256K of contiguous memory starting at zero:

```
MODULE  "MEMTBL"          ;module ident
        LOC   Data#       ;data segment
MEMTBL::                  ;memory spec table
        BYTE  1           ;just one region
        WORD  0x40        ;base (paragraph)
        WORD  0x4000-0x40   ;length (para)
        END
```

Note that the first 0x40 paragraphs (1K bytes) are reserved for 8086 interrupt vectors and must not be included in MEMTBL.

Console Driver

A console driver should be labelled with the public entry symbol CONDR_. A console number (from CONAST) is passed in register CH. The driver must perform a console I/O operation according to the operation code passed in register DL:

DL=	Function
0	Return status in AL, char in CL
1	Return input character in AL
2	Output character passed in CL
8	Enter error-message mode
9	Exit error-message mode
10	Conditional output char in CL

If DL=0, the driver determines if a console input character is available. If no character is available, the driver returns AL=0. If an input character is available, the driver returns AL=-1 and the input character in CL, but must not "consume" the character. TurboDOS depends upon this look-ahead capability to detect attention requests. The driver must not dispatch (via WAIT# or DELAY#) when processing a DL=0 call.

If DL=1, the driver returns an input character in AL (waiting if necessary).

If DL=2, the driver displays the output character passed in CL (waiting if necessary).

If DL=8, the driver prepares to display a TurboDOS error message; if DL=9, it reverts to normal. TurboDOS always precedes each error message with an DL=8 call and follows it with an DL=9 call. This gives the driver an opportunity to take special action (25th line, reverse video, etc.) for error messages. For simple consoles, the driver should output CR-LF in response to DL=8 or 9.

Console Driver (Continued)

If DL=10, the driver determines whether or not it can accept a console output character without dispatching (via WAIT# or DELAY#). If so, it outputs the character passed in CL, and returns AL=-1 to indicate that the character was accepted. However, if the driver cannot accept a console output character without dispatching, it returns AL=0 to indicate that the character was not accepted; TurboDOS will then make an DL=2 call to output the same character. This special conditional output call is used by TurboDOS to optimize console output speed by avoiding certain dispatch-related overhead whenever possible.

You should make a special effort to code the console driver to execute the minimum number of instructions possible, especially functions 0, 2, and 10. Excessive use of subroutine calls, stack operations, and other time-consuming coding techniques can make the difference between running the console device at full rated speed or something less.

Printer Driver

A printer driver should be labelled with the public entry symbol LSTDR_. A printer number (from PTRAST) is passed in register CH. The driver must perform a printer output operation according to the operation code passed in register DL:

DL=	Function
2	Print character passed in CL
7	Perform end-of-print-job action

If DL=2, the driver prints the output character passed in CL (waiting if necessary).

If DL=7, the driver takes any appropriate end-of-print-job action. This is quite hardware-dependent, and may include slewing to top-of-form, homing the print head, dropping the ribbon, and so forth.

Disk Driver

A disk driver should be labelled with the public entry symbol DSKDR_. The driver performs the physical disk operation specified by the Physical Disk Request (PDR) packet whose address is passed by TurboDOS in index register SI. The structure of the PDR packet is:

Offset	Contents		
;physical disk request (PDR) packet			
0[SI]	BYTE	OPCODE	;operation code
1[SI]	BYTE	DRIVE	;drive (base 0)
2[SI]	WORD	TRACK	;track (base 0)
4[SI]	WORD	SECTOR	;sector (base 0)
6[SI]	WORD	SECCNT	;#sectors to rd/wr
8[SI]	WORD	BYTCNT	;#bytes to rd/wr
10[SI]	WORD	DMAOFF	;DMA offs to rd/wr
12[SI]	WORD	DMABAS	;DMA base to rd/wr
14[SI]	WORD	DSTADR	;DST address
;copy of disk specification table (DST)			
16[SI]	BYTE	BLKSIZ	;block size (3-7)
17[SI]	WORD	NMBLKS	;#blocks on disk
19[SI]	BYTE	NMBDIR	;#directory blocks
20[SI]	BYTE	SECSIZ	;sector size (0-7)
21[SI]	WORD	SECTRK	;sectors per track
23[SI]	WORD	TRKDSK	;tracks on disk
25[SI]	WORD	RESTRK	;reserved tracks

The operation to be performed by the driver is specified in the first byte of the PDR packet (OPCODE) as follows:

OPCODE	Function
0	Read sectors from disk
1	Write sectors to disk
2	Determine disk type, return DST
3	Determine if drive is ready
4	Format track on disk

Disk Driver (Continued)

If OPCODE=0, the driver reads SECCNT physical sectors (or equivalently, BYTCNT bytes) into DMAOFF/DMABAS, starting at TRACK and SECTOR on DRIVE. The driver returns AL=0 if the operation is successful, or AL=-1 if an unrecoverable error occurs. TurboDOS may request multiple consecutive sectors to be read, but will never request an operation that extends past the end of the track.

If OPCODE=1, the driver writes SECCNT physical sectors (or BYTCNT bytes) from DMAOFF/DMABAS, starting at TRACK and SECTOR on DRIVE. The driver returns AL=0 if the operation is successful, or AL=-1 if an unrecoverable error occurs. TurboDOS may request multiple consecutive sectors to be written, but will never request an operation that extends past the end of the track.

If OPCODE=2, the driver must determine the type of disk mounted in DRIVE, and must return, in the DSTADR field of the PDR packet, the address of an 11-byte disk specification table (DST) 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

The first byte of the DST (BLKSIZ) specifies the allocation block size in bits 2-0. In addition, bit 7 is set if the disk is fixed (non-removable), and bit 6 is set if file extents are limited to 16K (EXM=0).

Disk Driver (Continued)

The driver returns AL=-1 if the operation is successful, or AL=0 if the drive is not ready or the disk type is unrecognizable. On successful return, TurboDOS moves a copy of the DST into 16[SI] through 26[SI], where it is available for subsequent operations.

If OPCODE=3, the driver determines whether DRIVE is ready, and returns AL=-1 if it is ready or AL=0 if not.

If OPCODE=4, the driver formats (initializes) TRACK on DRIVE, using hardware-dependent formatting information at DMAOFF/DMABAS (put there by the FORMAT command). The driver returns AL=0 if successful, or AL=-1 if an unrecoverable error occurs.

Network Driver

A network circuit driver should be labelled with the public entry symbol CKTDR_. A message buffer address is passed in register DX. The driver must either send or receive a network message, according to the operation code passed in register CL:

CL=	Function
0	Receive message into buffer at DX
1	Send message from buffer at DX

If CL=0, the driver receives a network message into the message buffer whose address is passed in DX (waiting if necessary). If a message is received successfully, the driver returns AL=0. If an unrecoverable malfunction of any remote processor is detected, the driver returns AL=-1 with the network address of the crashed processor in DX.

If CL=1, the driver sends a network message from the message buffer whose address is passed in DX. If the message is sent successfully, the driver returns AL=0. If the message could not be sent because of an unrecoverable malfunction of the destination processor, the driver returns AL=-1 with the network address of the crashed processor in DX.

The structure of a network message buffer is shown on the next page. The first four bytes of the buffer are reserved for a linkage used by TurboDOS, and should be ignored by the driver. The 11-byte message header and variable-length message body should be sent or received over the circuit. The driver should only need to look at the first two header fields (MSGLEN and MSGDID).

Network Driver (Continued)

```
; message buffer format
        WORD  ?         ;linkage (ignored)
        WORD  ?         ;   "        "
; 11-byte message header
        BYTE  MSGLEN    ;msg length
        WORD  MSGDID    ;destination addr
        BYTE  MSGPID    ;process id
        WORD  MSGSID    ;source addr
        WORD  MSGOID    ;originator addr
        BYTE  MSGOPR    ;orig'r process id
        BYTE  MSGLVL    ;forwarding level
        BYTE  MSGFCD    ;msg format code
; variable-length body
        RES   7         ;registers
        RES   38        ;optional FCB data
        RES   128       ;optional record
```

The length field MSGLEN represents the number of bytes in the message, including the header and body (but excluding the linkage). On a receive request (CL=0), TurboDOS presets MSGLEN to the maximum allowable message length, and expects MSGLEN to contain the actual message length on return. On a send request (CL=1), TurboDOS presets MSGLEN to the actual length of the message to be sent.

In a server/user network, it is often desirable for the circuit driver in the server to periodically "poll" the user processors on the circuit to detect any user malfunctions quickly and to effect recovery. If the driver reports that a user has crashed (by returning AL=-1 and DX=network-address), then the circuit driver must not accept any further messages from that user until TurboDOS has completed its recovery process.

Network Driver (Continued)

TurboDOS signals the driver that such recovery is complete by sending a dummy message destined for the user in question with a length of zero. The driver should not actually send such a message to the user, but could initiate whatever action is appropriate to reset the user and download a new copy of the user operating system.

A user must request an operating system download by sending a special download request message to the server (usually done by a bootstrap routine). The download request message consists of a standard 11-byte header (with MSGPID, MSGOID and MSGFCD zeroed) followed by a 1-byte body containing a "download suffix" character. The server processor addressed by MSGDID will return a reply message whose 128-byte body is the first record of the download file OSUSER-x.SYS (where "x" is the specified download suffix).

The user continues to send download request messages and to receive successive download records until it receives a short reply message (1-byte body) signifying end-of-file. The single byte passed as the body of the final short message identifies the system disk, and should be passed to the system in register AL.

The entire failure detection, failure recovery, and user downloading procedure is very hardware-dependent.

Comm Driver

The comm driver supports the TurboDOS communications extensions (T-functions 34-40), and may be omitted if these functions are not used. The driver should be labelled with the public entry symbol COMDRV. A comm channel number is passed in register CH. The driver must perform an I/O operation according to the operation code passed in register DL:

DL=	Function
0	Return input status in AL
1	Return input character in AL
2	Output character passed in CL
3	Set channel baud rate from CL
4	Return channel baud rate in AL
5	Set modem controls from CL
6	Return modem status in AL

If DL=0, the driver determines if an input character is available. If one is available, the driver returns AL=-1, otherwise AL=0.

If DL=1, the driver returns an input character in AL (waiting if necessary).

If DL=2, the driver outputs the character passed in CL.

If DL=3, the driver sets the channel baud rate according to the baud-rate code passed in CL. If DL=4, the driver returns the channel baud-rate code in AL. See T-functions 37 and 38 in the 8086 Programmer's Guide for baud-rate code definitions.

If DL=5, the driver sets the modem controls according to the bit-vector passed in CL. If DL=6, the driver returns the modem status vector in AL. See T-functions 39 and 40 in the 8086 Programmer's Guide for bit-vector definitions.

Clock Driver

The real-time clock driver does not take the form of a subroutine called by TurboDOS, as do the other drivers described in this section. Rather, the clock driver generally consists of an interrupt service routine which responds to interrupts from a periodic interrupt source (preferably 50 to 60 times a second). The interrupt service routine should call DLYTIC# once per system tick (to synchronize DELAY# requests). It should also call RTCSEC# once per second (that is, every 50 to 60 ticks) to update the system time and date. Finally, it should exit by jumping to ISRXIT# to provide a periodic dispatcher time-slice. Excluding initialization code, a typical clock driver might be coded thus:

```
RTCCNT: BYTE  60        ;divide-by-60 cntr
RTCISR: PUSH  AX        ;save registers
        PUSH  BX        ;  "       "
        PUSH  CX        ;  "       "
        PUSH  DX        ;  "       "
        PUSH  DS        ;  "       "
        CALL  GETSDS#   ;get system DS
        CALL  DLYTIC#   ;signal one tick
        DEC   RTCCNT    ;decrement counter
        JNZ   __X       ;not 60 ticks yet
        MOV   RTCCNT,=60  ;reset counter
        CALL  RTCSEC#   ;signal one second
__X:    MOV   DX,&EOIR  ;DX=&end-of-int
        MOV   AX,=INTN  ;AX=interrupt#
        OUT   DX,AX     ;reset interrupt
        POP   DS        ;restore registers
        POP   DX        ;   "        "
        POP   CX        ;   "        "
        POP   BX        ;   "        "
        POP   AX        ;   "        "
        JMP   ISRXIT#   ;go to dispatcher
```

Clock Driver (Continued)

If the hardware is capable of determining the date and time-of-day at cold-start (by means of a battery-powered clock, for example), the clock driver may initialize the following public symbols in the RTCMGR module:

```
SECS::  BYTE  0       ;seconds 0-59
MINS::  BYTE  0       ;minutes 0-59
HOURS:: BYTE  0       ;hours   0-24
JDATE:: WORD  0x8001  ;Julian date
                      ;base 31-Dec-47
```

Bootstrap

The bootstrap is usually contained in a ROM or on a boot track. Its function is to search all disk drives for the TurboDOS loader program OSLOAD.CMD, and to load and execute it if found. To generate a bootstrap, use TLINK to combine the standard bootstrap module OSBOOT.O with your own hardware-dependent driver. Your driver must define the following public names: INIT, SELECT, READ, XFER, CODE, and DATA.

INIT:: is called once to perform any required hardware initialization. It returns with register AX set to the paragraph address of the load base (where the file OSLOAD.CMD should be loaded into memory by the bootstrap). This address should be chosen so that OSLOAD will not overlay the bootstrap or the operating system to be loaded.

SELECT:: is called to select the disk drive passed in AL (0-15). If the selected drive is not ready or non-existent, it returns AL=0. Otherwise, it returns AL=-1 and the address of an 11-byte disk specification table (DST) in register SI (see page 5-7).

READ:: is called to read one physical sector from the last-selected drive. The track is passed in CX, the sector in DX, the DMA offset in BX, and the DMA base in ES. It must return AL=0 if successful, or AL=-1 if an unrecoverable error occurred.

XFER:: is transferred to at the end of the bootstrap process. In most cases, this routine must set register DS to the base paragraph address of the loader (normally the load base returned by INIT:: plus 8 to allow for the .CMD header), set location DS:0080 to zero (to simulate a null command tail), and jump to the loader (using a JMPF to set CS=DS and IP=0x100).

Bootstrap (Continued)

CODE:: defines the base paragraph (CS value) under which the bootstrap itself is to be executed. OSBOOT loads this value into register CS before calling INIT::, SELECT::, READ:: or XFER::.

DATA:: defines the base paragraph (DS value) of a 128-byte RAM area that OSBOOT may use for working storage. (It should not be located where OSLOAD.CMD will be loaded!) OSBOOT loads this value into register DS before calling INIT::, SELECT::, READ:: or XFER::.

OTOASM Command

Some TurboDOS implementations require that a Z80 server processor download 8086-family user processors. In writing the network circuit driver for the Z80 server processor, it is often necessary to embed a download bootstrap routine written in 8086 code. The utility program OTOASM.CMD is designed to simplify this process.

OTOASM converts an 8086 object file (type .O) produced by TASM into a Z80 source file (type .ASM) acceptable to either the PASM or M80 assemblers. The output file contains a sequence of data definition statements (.BYTE and .WORD, or DB and DW) representing 8086 machine-language.

Syntax

```
OTOASM  filename  {-M}
```

Explanation

The "filename" argument must not have an explicit type, and specifies the name of both the input file "filename.O" and the output file "filename.ASM" to be used. The "-M" option causes the output to be formatted for the M80 assembler rather than the PASM assembler.

The input file (type .O) must not contain any relocatable tokens. Consequently, the 8086 source module (type .A) must define only absolute location counter values (LOC) and must make no external references (# suffix). Public symbols may be defined as long as they do not have relocatable values.

(Intentionally left blank.)

User OS Patch Points

The following User OS Patch Points are supported.

Patch Point	Description
CONBR	Baud rate patch point in module CON96. Default = 9600-0xCE.

Baud Rate Code:

bit 7 = 1 if attention detection is enabled
bit 6 = 1 if clear-to-send handshaking enabled
bit 5 = 1 if output-only (input disabled)
bits 3-0 = baud-rate value 0..15 (see table below)

Notes: The least significant nibble of the E-register contains a baud rate value as follows:

0 =	50	8 =	1,800
1 =	75	9 =	2,000
2 =	110	10 =	2,400
3 =	134.5	11 =	not used
4 =	150	12 =	4,800
5 =	300	13 =	7,200
6 =	600	14 =	9,600
7 =	1,200	15 =	19,200

CTSBR	Baud rate patch point, in LSTCTS module (see list above). Default = 9600 = 0x4E.
E1XBR	Baud rate patch point, in LSTETX module (see list above). Default = 1200 = 0x47.
ETXLEN	Block length prior to ETX signal. Default = 0x6E.

Patch Points	Description
XONBR	Baud rate patch point in LSTXON module (see list above).

Server OS Patch Points

The following Server OS Patch Points are supported.

Patch Points	Description
NSMTOP	Top of physical memory, in MPEHRM module. Default - 0FFFF.
NSFTOP	Top of memory above floppy controller, in MPEHRM module. Default = 0F000.

Note: MPEHRM releases RAM from NSFTOP to NSMTOP to the TurboDOS memory pool.

The following are all in the MCDU16 module:

CKTU16	HRZ-UP16 board circuit number. Default = 0.
NMBU16	Number of HRZ-UP16's supported. Default = (set by CONFIG).
SSTU16	Suffix table for User OS. Default = "BBBBBBBB"
PATU16	I/O port addresses for HRZ-UP16s. Defaults = 40, 42, 44, 46, 48, 4A, 4C, 4E.