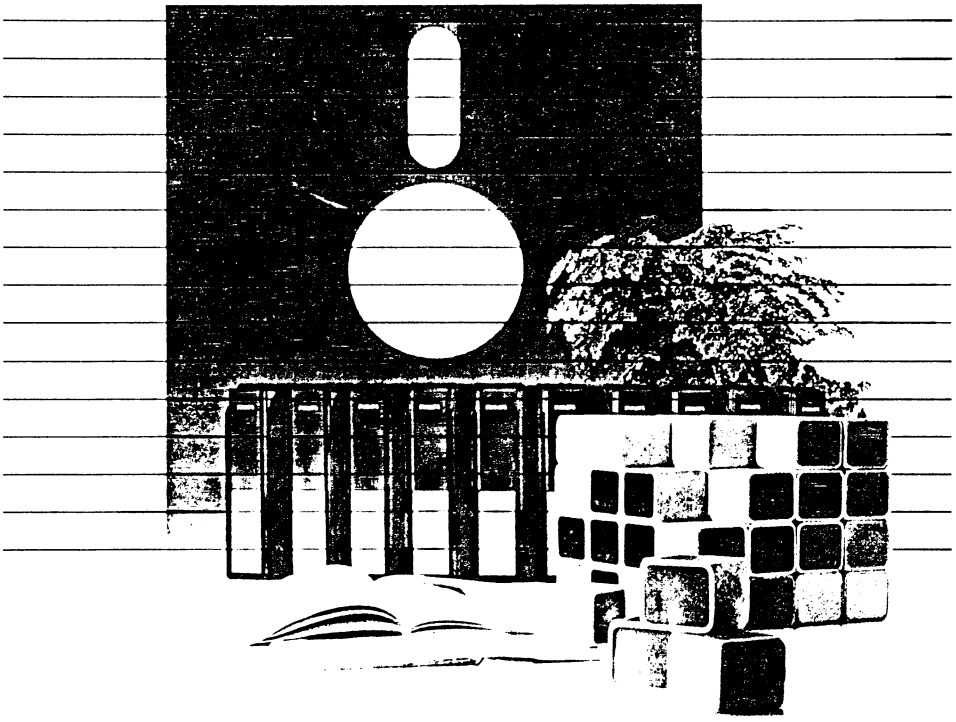
P3500/P3800
TurboDOS 1.4
8086 implementor's guide (16 bits)



PHILIPS

July 1984

Order Number 5122 993 81839

Manual Number F77H

TurboDOS 1.4

8086 Implementor's Guide

June 1984

Software 2000, Inc.
1127 Hetrick Avenue
Arroyo Grande, CA 93420
U.S.A.

Copyright Notice

Trademark Notice

TurboDOS is a registered trademark of Software 2000, Inc., and has been registered in the United States and in most major countries of the free world.

IBM is a trademark of International Business Machines Corporation. CP/M, Concurrent CP/M and MP/M are trademarks of Digital Research.

Disclaimer

Software 2000, Inc., makes no representations or warranties with respect to the contents of this publication, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Software 2000, Inc., shall under no circumstances be liable for consequential damages or related expenses, even if it has been notified of the possibility of such damages.

Software 2000, Inc., reserves the right to revise this publication from time to time without obligation to notify any person of such revision.

First Edition: June 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, and includes assembler source listings of sample drivers.

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 1.4 User's Guide
- TurboDOS 1.4 8086 Programmer's Guide
- TurboDOS 1.4 Z80 Programmer's Guide
- TurboDOS 1.4 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 master
- simple network slave (no local disks)
- complex network slave (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 various 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.

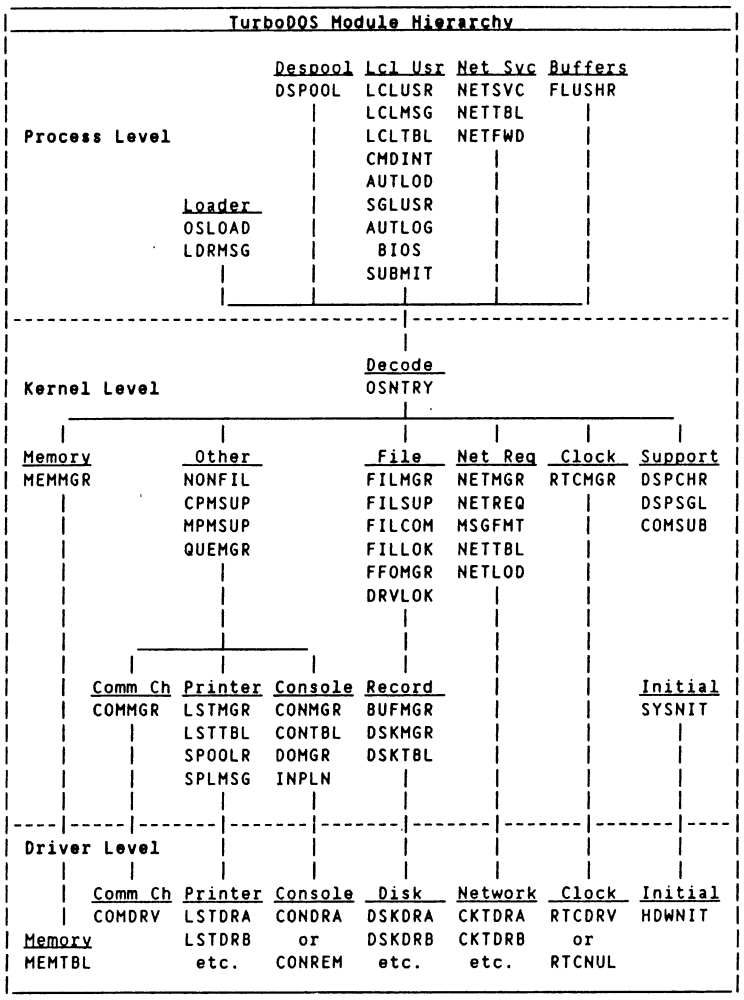
Driver Level

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.CMD 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 (OSMASTER.SYS) into memory at each system cold-start.



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.
SGLUSR	Flushes disk buffers at each console input. Use for single-user systems instead of FLUSHR.
AUTLOG	Automatic log-on routine. Used when full log-on security is not desired. See AUTUSR patch point.
BIOS	Direct BIOS Call (C-fcn 50).
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.
NETFWD	Manages network message forwarding. Requires NETREQ+NETSVC.
DSPOOL	Processes background printing.
FLUSHR	Periodically flushes disk buffers. Use for network master 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
MPMSUP	Processes C-functions 141-143, 153, 160, 161 (optional).
QUEMGR	Emulates MP/M queues, supports C-functions 134-140 (optional). Requires MPMSUP.
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.

Kernel Modules (Continued)

Module	Function
NETLOD	Loads programs over the network.
RTCMGR	Real-time clock manager keeps system date and time.
DSPCHR	Multi-task dispatcher which controls sharing of the local processor among multiple processes.
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 master to support MASTER 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 master
STDSLAVE	simple slave w/o local disks
STDSLAVX	complex slave with local disks

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

Module	K	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
AUTLOD	.2	-	AUTLOD	AUTLOD	AUTLOD	AUTLOD	AUTLOD
AUTLOG	.0	-	AUTLOG	AUTLOG	AUTLOG	AUTLOG	AUTLOG
BIOS	.3	-	BIOS	BIOS	BIOS	BIOS	BIOS
BUFMGR	1.2	BUFMGR	BUFMGR	BUFMGR	BUFMGR	-	BUFMGR
CMDINT	1.7	-	CMDINT	CMDINT	CMDINT	CMDINT	CMDINT
COMMGR	.1	-	COMMGR	COMMGR	COMMGR	COMMGR	COMMGR
COMSUB	.2	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB
CONMGR	.4	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR
CONREM	.5	-	-	-	+	-	-
CONTBL	.0	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL
CPMSUP	.3	-	+	+	+	+	+
DOMGR	.4	-	DOMGR	DOMGR	DOMGR	DOMGR	DOMGR
DRVLOK	.1	-	-	-	DRVLOK	-	-
DSKMGR	.6	DSKMGR	DSKMGR	DSKMGR	DSKMGR	-	DSKMGR
DSKTBL	.0	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL
DSPCHR	.7	-	-	DSPCHR	DSPCHR	DSPCHR	DSPCHR
DSPOOL	1.0	-	-	DSPOOL	DSPOOL	-	DSPOOL
DSPSGL	.2	DSPSGL	DSPSGL	-	-	-	-
FFOMGR	1.1	-	-	-	FFOMGR	-	-
FILCOM	.4	FILCOM	FILCOM	FILCOM	FILCOM	FILCOM	FILCOM
FILLOK	2.0	-	-	-	FILLOK	-	-
FILMGR	2.5	FILMGR	FILMGR	FILMGR	FILMGR	-	FILMGR
FILSUP	2.9	FILSUP	FILSUP	FILSUP	FILSUP	-	FILSUP
FLUSHR	.2	-	-	-	FLUSHR	-	-
INPLN	.2	-	INPLN	INPLN	INPLN	INPLN	INPLN
LCLMSG	.4	-	LCLMSG	LCLMSG	LCLMSG	LCLMSG	LCLMSG
LCLTBL	.0	-	LCLTBL	LCLTBL	LCLTBL	LCLTBL	LCLTBL
LCLUSR	1.1	-	LCLUSR	LCLUSR	LCLUSR	LCLUSR	LCLUSR
LDRMSG	.1	LDRMSG	-	-	-	-	-
LSTMGR	.3	-	LSTMGR	LSTMGR	LSTMGR	LSTMGR	LSTMGR
LSTTBL	.1	-	LSTTBL	LSTTBL	LSTTBL	LSTTBL	LSTTBL
MEMMGR	1.2	-	MEMMGR	MEMMGR	MEMMGR	MEMMGR	MEMMGR
MPMSUP	.1	-	+	+	+	+	+
MSGFMT	.1	-	-	-	+	MSGFMT	MSGFMT
NETFWD	.3	-	-	-	+	+	+
NETLOD	.3	-	-	-	+	NETLOD	NETLOD
NETMGR	.9	-	-	-	NETMGR	NETMGR	NETMGR
NETREQ	1.6	-	-	-	+	NETREQ	NETREQ
NETSVC	1.8	-	-	-	NETSVC	+	+
NETTBL	.0	-	-	-	NETTBL	NETTBL	NETTBL
NONFIL	.2	NONFIL	NONFIL	NONFIL	NONFIL	NONFIL	NONFIL
OSLOAD	1.1	OSLOAD	-	-	-	-	-

Module	K	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
OSNTRY	.5	OSNTRY	OSNTRY	OSNTRY	OSNTRY	OSNTRY	OSNTRY
PATCH	.1	+	+	+	+	+	+
PGMLOD	1.0	-	PGMLOD	PGMLOD	PGMLOD	PGMLOD	PGMLOD
QUEMGR	1.3	-	-	-	+	+	+
RTCMGR	.1	-	RTCMGR	RTCMGR	RTCMGR	-	RTCMGR
RTCNUL	.1	+	+	+	+	+	+
SGLUSR	.1	-	SGLUSR	SGLUSR	-	-	SGLUSR
SPLMSG	.1	-	-	SPLMSG	SPLMSG	SPLMSG	SPLMSG
SPOOLR	.6	-	-	SPOOLR	SPOOLR	SPOOLR	SPOOLR
SUBMIT	.2	-	+	+	+	+	+
SYSNIT	.1	-	SYSNIT	SYSNIT	SYSNIT	SYSNIT	SYSNIT

Optional Modules

To supplement the standard packages, certain optional modules (marked by "+" in the matrix above) may have to be added. The following table explains where these optional modules are required:

Module	Where Required
CONREM	Network masters with no console (instead of CONDR_).
CPMSUP	To support C-fcns 7, 8, 24, 28, 29, 31, 37 and 107.
MPMSUP	To support C-fcns 134-143, 153, 160 and 161.
MSGFMT	Network masters that make requests over the network.
NETFWD	To support forwarding of network messages.
NETLOD	Network masters that load programs over the network.
NETREQ	Network masters that make requests over the network.
PATCH	Wherever a supplementary patch area is required.
QUEMGR	To support MP/M queue emulation (C-fcns 134-140.)
RTCNUL	Wherever no RTC driver is available.
SUBMIT	To emulate CP/M processing of $$$.SUB.

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 master. 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 masters.

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

	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
O/S	10K	17K	19K	25K	13K	22K
Drivers	2K	2K	2K	3K	1K	2K
Buffers	4K	4K	4K	16K	-	4K
Dynamic	1K	1K	1K	3K	2K	2K
Total	17K	24K	26K	47K	16K	30K

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 OEM 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 both a loader OSLOAD.CMD and a master operating system OSMASTER.SYS. For a networking system you also have to generate a slave operating system OSSLAVE.SYS. Complex networks may require generation of several different slave or master 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.CMD into the TPA of the master computer and executes it. OSLOAD loads the master operating system from the file OSMASTER.SYS into memory. The master operating system then down-loads the slave operating system from the file OSSLAVE.SYS over the network into each slave 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. Patches must be stored in a .PAR file. 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 (...) or brackets [...]; otherwise, it is stored as an 8-bit byte. An expression enclosed in brackets is treated as IP-relative (for example, the target address of a CALL or JMP instruction). 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 TASM backslant escape sequences.

Example

In the following example, TLINK is used to link a single-user TurboDOS system for an IBM Personal Computer, using the modules listed in OSMASTER.GEN and patches in OSMASTER.PAR, creating the executable file OSMASTER.SYS.

```
0A}TLINK OSMASTER.SYS -M
Copyright 1984, Software 2000, Inc.
* ; Single-user without spooling for
* ; IBM Personal Computer with 256K RAM
* STDSINGL  ;standard single-user pkg.
* CPMSUP    ;seldom-used CP/M functions
* CONIPC    ;IBM PC console driver
* LSTACA    ;IBM PC serial list driver
* NITIPC    ;IBM PC initialization
* DSKIPC    ;IBM PC floppy disk driver
* MSTIPC    ;IBM PC 256K mem spec table
* RTCIPC    ;IBM PC real-time clock drvr

Pass 1
LCLUSR LCLTBL CMDINT AUTLOD SGLUSR etc.

Pass 2
LCLUSR LCLTBL CMDINT AUTLOD SGLUSR etc.

Processing parameter file:
; Patches for single-user w/o spooling
OSMLEN = 1024 ;dynamic memory area (16K)
OSMTOP = 0x1000  ;but limit to first 64K
AUTUSR = 0x80 ;logon to user 0 privileg.
NMBUFS = 8    ;number of disk buffers
EOPCHR = 0x1A ;end-of-print character ^Z
SRHDRV = 1    ;search drive A
PRTMOD = 0    ;direct printing mode

Writing output file A:OSMASTER.SYS
0A}
```

Error Messages

```
Serial number violation
Not enough memory
No object files specified
Can't open object file
Non-privileged user
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 various 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).		
BUFBAS = (0000)		BUFMGR
Base paragraph address of external disk buffer area (see BUFLEN).		
BUFLEN = (0000)		BUFMGR
Length (in paragraphs) of external disk buffer area starting at BUFLEN. Default value (0000) indicates that buffers are to be allocated from the regular dynamic memory pool (see OSMLEN, OSMTOP).		
BUFSIZ = 3		BUFMGR
Default disk buffer size (0=128, 1=256, 2=512, 3=1K,..., 7=16K). Default value specifies 1K disk buffers.		

Patch Points
(Continued)

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.)		
CLBLEN = 157		CMDINT
Command line buffer length defines longest permissible command line. The default value permits two 80-char lines.		
CLPCHR = "}"		CMDINT
Command line prompt character.		
CLSCHR = "\\"		CMDINT
Command line separator character.		
COLDFN = 0,"COLDSTRT","AUT"		AUTLOD
File name and drive for cold-start autoload processing (in FCB format).		

Patch Points (Continued)

Symbol	Default Value	Module
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 BL-register.		
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 slave, DEFDID should be set to the network address of the master.		

Patch Points (Continued)

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). NOTE: In slave configurations STDSLAVE and STDSLAVX, the default values are: DSKAST = 0x80,(0),0x81,(0), 0x82,(0),0x83,(0), ...,0x8E,(0),0x8F,(0)	

Patch Points
(Continued)

Symbol	Default Value	Module
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.	
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.	

Patch Points (Continued)

Symbol	Default Value	Module
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.		
LOGUSR = 31		FILCOM
User number for logged-off state.		
MAXMBS = 0		NETMGR
Maximum number of message buffers that will ever be allocated. Default value of 0 means number of message buffers is limited only to size of available memory.		

Patch Points (Continued)

Symbol	Default Value	Module
MAXRPS = 0		NETMGR
Maximum number of reply packets that will ever be allocated. Default value of 0 means number of reply packets is limited only to the size of available memory.		
NMBCKT = 1		NETTBL
Number of network circuits to which this processor is connected.		
NMBMBS = 0		NETMGR
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).		
NMBRPS = 0		NETMGR
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).		

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 dynamic working storage. This area must accomodate disk buffers if no external disk buffer area is defined (BUFLEN is zero). The default value (128 paragraphs or 2K bytes) is appropriate for a simple slave with no disk buffers. For other configurations, patch OSMLEN to a value large enough to accomodate dynamic memory needs. Divide required length in bytes by 16 to give the value of OSMLEN in paragraphs. (See OSMTOP.)		

Patch Points (Continued)

Symbol	Default Value	Module
OSMTOP = (0000)		MEMMGR
Absolute upper bound (paragraph address) for dynamic working storage area. The actual upper bound is either OSMTOP or the top of TurboDOS plus OSMLEN, whichever is smaller. The default value of zero indicates no specified upper bound.		
PRTCHR = 0x0C ;CTRL-L		CONTBL
End-print character (after attention). This is a console attention-response, not to be confused with EOPCHR.		
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.		

Patch Points (Continued)

Symbol	Default Value	Module
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 slave 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 slave configurations STDSLAVE and STDSLAVX, the default values are: QUEAST = 0x80,(0),0x81,(0), 0x82,(0),0x83,(0), ...,0x8E,(0),0x8F,(0)		
QUEDLY = (0000)		QUEMGR
Polling delay used in unconditional Read Queue (when queue is empty) and Write Queue (when queue is full), stated in system "ticks". If RTC driver is available, patch to largest delay that yields reasonable queue performance.		

Patch Points (Continued)

Symbol	Default Value	Module
QUEDRV = 0xFF		QUEMGR
Drive used for FIFOs that emulate MP/M queues. Default value 0xFF means use the system disk (disk from which TurboDOS was loaded at cold-start). Patch to 0 - 15 to specify a particular drive A-P.		
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.		
RCNMSK = 0xFF		MPMSUP
Mask used in deriving a console number from a network node in C-function 153.		
RCNOFF = 0		MPMSUP
Offset used in deriving a console number from a network node in C-function 153.		
RESCHR = 0x11 ;CTRL-Q		CONTBL
Resume character (after attention).		

Patch Points (Continued)

Symbol	Default Value	Module
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 slave processors.		
SPLDRV = 0xFF		LCLTBL
Initial spool drive. Default value 0xFF indicates spool to system disk (disk from which TurboDOS was loaded at cold-start). Patch to 0 - 15 to specify 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.		
SUBFN = 0,"$$$ ","SUB"		SUBMIT
FCB for emulating CP/M submit files.		
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 master/slave 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)

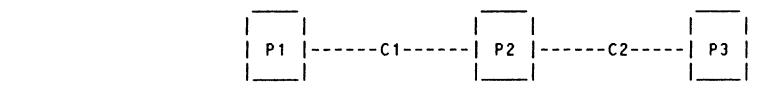
Symbol	Description
FWDTBL	The message forwarding table that specifies any additional circuits (not directly connected to this processor) which may be accessed via explicit message forwarding, and how messages destined for such circuits are to be routed.

These tables are pre-defined with default values to make set-up of simple master/slave 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 sub-section for details of the organization and defaults for these network tables.

Message Forwarding

The TurboDOS module NETFWD supports both "implicit" and "explicit" forwarding of network messages. 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:



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.

A Complex Example

Let's take a reasonably complex network situation and see how to construct the required .GEN and .PAR files.

Our hardware is a board-and-bus microcomputer system consisting of an 80286 CPU running in unmapped (8086) mode, 128K of RAM, hard disk and floppy disk subsystems (all these make up the master processor), and several single-board slave computers with 80186 CPUs and 256K of RAM each. The master processor is interfaced to two printers via RS232 serial ports: a daisywheel printer on port 0 using XON/XOFF protocol and a matrix printer on port 1 using clear-to-send handshaking. In addition, the master has a high-speed RS422 interface connecting it to another board-and-bus system of similar configuration some distance away.

We want to configure a TurboDOS system for this hardware that permits all of the users of each system to access the hard disk, floppy disks, and printers attached to both the local and remote system. We might create the following OSMASTER.GEN file:

```
; OSMASTER.GEN for complex example
STDMASTR ; standard master package
NETREQ   ; to make requests of other sys
MSGFMT   ; needed by NETREQ
CONREM   ; no console on the master
LSTXON   ; XON/XOFF for daisy    (LSTDRA)
LSTCTS   ; CTS for matrix        (LSTDRB)
DSKHDC   ; hard disk controller  (DSKDRA)
DSKFDC   ; floppy disk control.  (DSKDRB)
CKTSLV   ; circuit driver for slaves (C0)
CKT422   ; circuit driver for RS422  (C1)
RTCDRV   ; real-time clock driver
NITDRV   ; hardware initialization driver
MEMTBL   ; memory specification table
```

A Complex Example (Continued)

Our system generation task is completed by creating the companion OSMASTER.PAR file:

```
; OSMASTER.PAR for complex example
NMBCKT = 2              ; 2 network circuits:
CKTAST = (0x0000),(CKTDRA) ; C0 = bus
         (0x0100),(CKTDRB) ; C1 = RS422
DSKAST = 0x00,(DSKDRA) ; drv A=local HD
         0x00,(DSKDRB) ; drv B=local FD0
         0x01,(DSKDRB) ; drv C=local FD1
         0x80,(0x0101) ; drv D=remote HD
         0x81,(0x0101) ; drv E=remote FD0
         0x82,(0x0101) ; drv F=remote FD1
PTRAST = 0x00,(LSTDRA) ; ptr A=lcl daisy
         0x01,(LSTDRB) ; ptr B=lcl matrix
         0x80,(0x0101) ; ptr C=rmt daisy
         0x81,(0x0101) ; ptr D=rmt matrix
QUEAST = 0x00,(0x0000) ; queue A=local
         0x00,(0x0000) ; queue B=local
         0x80,(0x0101) ; queue C=remote A
         0x81,(0x0101) ; queue D=remote B
DEFDID = (0x0101)   ; default=other master
DSPPAT = 1,2,3,4    ; assgn ptrs to queues
OSMLEN = (0x0600)   ; 24K dynamic memory
COMPAT = 0x88       ; compatibility flags
NMBSVC = 5          ; 5 server processes
NMBUFS = 20         ; 20 1K disk buffers
```

The generation of the second master operating system could be identical, except that all occurrences of network addresses (0x0100) and (0x0101) in the OSMASTER.PAR file would be reversed. Generation of the slave operating system would be very straightforward, and identical for both systems.

If you study this example thoroughly until you understand the reason for every .GEN and .PAR file entry, you should have little trouble setting up your own "sysgens".

Sysgen Procedure

To conclude this section, here is a suggested step-by-step procedure for generating a new version of TurboDOS:

1. Bring up a previous version of 8086 TurboDOS. If this is your first attempt to generate an 8086 TurboDOS system, you may bring up CP/M-86 instead. However, if you are using CP/M, all disks will have to be in a format compatible with both CP/M and TurboDOS (e.g., eight-inch one-sided single-density with 128-byte sectors).

2. Make a working copy of your TurboDOS distribution disk. Do not use the original disk (in case something goes wrong). Insert the working diskette in a convenient disk drive.

3. Using your favorite text editor, create or revise the file OSMASTER.GEN containing the names of the relocatable modules to be linked together. Generally, this will consist of the appropriate STDxxxxx standard package plus selected additional modules and all required device drivers.

4. Using your editor once again, create or revise the file OSMASTER.PAR containing any required patches. This may be omitted if no patches are desired.

5. Using the command TLINK OSMASTER.SYS, generate an executable master operating system in accordance with the .GEN and .PAR files.

6. In a similar fashion, construct a new loader by creating or revising the files OSLOAD.GEN and OSLOAD.PAR, then using the command TLINK OSLOAD.CMD to generate the executable loader.

Sysgen Procedure

7. For a master/slave network system, construct a slave operating system in the same manner. Create or revise the files OSSLAVE.GEN and OSSLAVE.PAR, then use the command TLINK OSSLAVE.SYS to generate the down-loadable slave operating system.

8. To test the newly-generated system, eject all disks other than your working disk (again, in case something goes wrong). Enter the command OSLOAD. The new system should cold-start. If it fails to come up or to function properly, you will have to start over at step 1 and check your work carefully -- there is most likely an error in one of your .GEN or .PAR files, or a "bug" in one of your drivers.

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 do not have to be licensed separately from the network master. (This would be the case, for example, if the slave computers have no local disk storage, or if TurboDOS is furnished in a form that cannot be run stand-alone on the slave computers.) However, networked computers that are also capable of stand-alone operation under TurboDOS must each be licensed separately.

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 reproduce TurboDOS programs or documentation for any purpose. 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 distribu ted 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 re-serialize 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
Non-privileged user
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 (1.4x)

   . 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
MPMSUP .O	MPMSUP .O	MPMSUP .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
-	-	NETFWD .O
-	-	QUEMGR .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

Distribution Procedure (Continued)

single-user w/o spooler	single-user with spooler	multi-user networking
ERASEDIR.CMD	ERASEDIR.CMD	ERASEDIR.CMD
-	-	FIFO .CMD
FIXDIR .CMD	FIXDIR .CMD	FIXDIR .CMD
FIXMAP .CMD	FIXMAP .CMD	FIXMAP .CMD
FORMAT .CMD	FORMAT .CMD	FORMAT .CMD
LABEL .CMD	LABEL .CMD	LABEL .CMD
-	-	LOGOFF .CMD
-	-	LOGON .CMD
-	-	MASTER .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
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 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. (At present, only the LSB of the count word is used, supporting counts in the range -128 to +127.) 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£, never by spinning in a loop.

Interrupt Service

Dispatching is especially efficient when used with interrupt-driven devices. Usually, the interrupt service routine just calls SIGNALE 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 ISRXITE to provide periodic time-slicing of processes. To avoid excessive dispatcher overhead, don't use ISRXITE more than about 60 times per second.

Before calling any TurboDOS support routine (such as SIGNALE) or referencing any DS-relative data, an interrupt service routine must call the subroutine GETSDSE 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  GETSDSE   ;get system DS
        MOV   BX,&EVENT  ;BX=&semaphore
        CALL  SIGNALE   ;signal event
        MOV   DX,&EOIR  ;DX=&end-of-int
        MOV   AX,=INTN  ;AX=interruptE
        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 WAITE upon entry and SIGNALE 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  WAITE      ;wait if in-use
          :

          :
        MOV   BX,&MXSPH  ;BX=&semaphore
        CALL  SIGNALE    ;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  WAITE      ;lock MX
        STI              ;need ints enabled
        MOV   BX,&RDASPH ;BX=&semaphore
        CALL  WAITE      ;wait data avail
        PUSH  CHRSAV     ;stack input char
        MOV   BX,&MXSPH  ;BX=&MXsemaphore
        CALL  SIGNALE    ;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  GETSDSE    ;get system DS
        IN    AL,=INPUT  ;get input char
        MOV   CHRSAV,AL  ;save for driver
        MOV   BX,&RDASPH ;BX=&semaphore
        CALL  SIGNALE    ;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 must not 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, ES: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 AL-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 WAITf or DELAYf. 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   Dataf     ;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 WAITf or DELAYf) 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 WAITf or DELAYf). 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. Study the sample driver listings in the appendix with this in mind.

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 two words 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 needs to look at only the first two header fields (MSGLEN and MSGDID) and possibly the last field (MSGFCD).

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   1         ;user f and flags
        RES   37        ;optional FCB data
        RES   128       ;optional record
```

The message format code field MSGFCD contains bit-encoded flags that define the format and context of each network message. This field may be ignored by most simple drivers, but its contents may be useful in complex network environments. Encoding of MSGFCD is:

Bit	Meaning
0	first message of session
1	last message of session
2	continuation message follows
3	request includes FCB data
4	request includes record data
5	reply includes FCB data
6	reply includes record data
7	this is a reply message

Network Driver (Continued)

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

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

A slave must request an operating system download by sending a special download request message to the master (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 master processor addressed by MSGDID will return a reply message whose 128-byte body is the first record of the download file OSSLAVEx.SYS (where "x" is the specified download suffix).

Network Driver (Continued)

The slave 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 slave downloading procedure is very hardware-dependent. Study the driver listing in the appendix for guidance.

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 master processor download 8086-family slave processors. In writing the network circuit driver for the Z80 master 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.

SAMPLE DRIVER SOURCE LISTINGS

The remainder of this document consists of assembler source listings of actual drivers. The listings comprise the drivers for a working TurboDOS system for the IBM Personal Computer with 256K of RAM.

The listings appear in the following order:

Module	Description
DREQUATE	common symbolic equates
MPBIPC	IBM PC bootstrap driver
NITIPC	IBM PC driver initialization
CONIPC	IBM PC TTY-mode console driver
LSTPPA	IBM PC parallel printer driver
LSTACA	IBM PC serial printer driver
RTCIPC	IBM PC real-time clock driver
DSKIPC	IBM PC floppy disk driver
MSTIPC	IBM PC memory spec table (256K)

Network circuit drivers will be furnished in the next edition of this document. In the meantime, refer to the Z80 Implementor's Guide for circuit driver examples.

```
; £INCLUDE      "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALENCES
        £NOLIST            ;SUPPRESS LISTING
;
; ASCII EQUIVALENCES
;
ANUL    == BYTE 0x00    ;NULL
ASOH    == BYTE 0x01    ;SOH
ASTX    == BYTE 0x02    ;STX
AETX    == BYTE 0x03    ;ETX
AEOT    == BYTE 0x04    ;EOT
AENQ    == BYTE 0x05    ;ENQ
AACK    == BYTE 0x06    ;ACK
ABEL    == BYTE 0x07    ;BELL
ABS     == BYTE 0x08    ;BS
AHT     == BYTE 0x09    ;HT
ALF     == BYTE 0x0A    ;LF
AVT     == BYTE 0x0B    ;VT
AFF     == BYTE 0x0C    ;FF
ACR     == BYTE 0x0D    ;CR
ASO     == BYTE 0x0E    ;SO
ASI     == BYTE 0x0F    ;SI
ADLE    == BYTE 0x10    ;DLE
ADC1    == BYTE 0x11    ;DC1
ADC2    == BYTE 0x12    ;DC2
ADC3    == BYTE 0x13    ;DC3
ADC4    == BYTE 0x14    ;DC4
ANAK    == BYTE 0x15    ;NAK
ASYN    == BYTE 0x16    ;SYN
AETB    == BYTE 0x17    ;ETB
ACAN    == BYTE 0x18    ;CAN
AEM     == BYTE 0x19    ;EM
ASUB    == BYTE 0x1A    ;SUB
AESC    == BYTE 0x1B    ;ESC
AFS     == BYTE 0x1C    ;FS
AGS     == BYTE 0x1D    ;GS
ARS     == BYTE 0x1E    ;RS
AUS     == BYTE 0x1F    ;US
ASP     == BYTE 0x20    ;SPACE
ARUB    == BYTE 0x7F    ;RUBOUT (DEL)
;
        LOC     0       ;SYMBOLIC DEFINITIONS RELATIVE TO 0
;
PDRDP:                  ;PD REQUEST DESCRIPTOR PACKET
PDRFCN: RES     BYTE 1  ;PD REQUEST FUNCTION NUMBER
PDRDRV: RES     BYTE 1  ;PD REQUEST DRIVE NUMBER
PDRTRK: RES     WORD 1  ;PD REQUEST TRACK NUMBER
PDRSEC: RES     WORD 1  ;PD REQUEST SECTOR NUMBER
PDRSC:  RES     WORD 1  ;PD REQUEST SECTOR COUNT
PDRTC:  RES     WORD 1  ;PD REQUEST TRANSFER COUNT
PDRDMA: RES     WORD 1  ;PD REQUEST DMA OFFSET
PDRBAS: RES     WORD 1  ;PD REQUEST DMA BASE
PDRDST: RES     WORD 1  ;PD REQUEST DRIVE SPEC TABLE ADDR
PDRLEN  == BYTE .-PDRDP ;PD REQUEST DESCRIPTOR PACKET LENGTH
DSKNFO:                 ;DISK TYPE INFORMATION
BLKSIZ: RES     BYTE 1  ;BLOCK SIZE
NMBLKS: RES     WORD 1  ;NUMBER OF BLOCKS
NMBDIR: RES     BYTE 1  ;NUMBER OF DIRECTORY BLOCKS
SECSIZ: RES     BYTE 1  ;PHYSICAL SECTOR SIZE (2^N*128)
SECTRK: RES     WORD 1  ;PHYSICAL SECTORS PER TRACK
TRKDSK: RES     WORD 1  ;PHYSICAL TRACKS PER DISK
```

```
RESTRK: RES     WORD 1  ;NUMBER OF RESERVED TRACKS
DNFOL   == BYTE .-DSKNFO  ;DISK INFO LENGTH
;
        RELOC           ;RESTORE PREVIOUS LOCATION
;
        £RELIST         ;RESTORE PREVIOUS LISTING STATUS
```

```
;
; ASCII EQUIVALENCES
;
ANUL    == 00H  ;NULL
ASOH    == 01H  ;SOH
ASTX    == 02H  ;STX
AETX    == 03H  ;ETX
AEOT    == 04H  ;EOT
AENQ    == 05H  ;ENQ
AACK    == 06H  ;ACK
ABEL    == 07H  ;BELL
ABS     == 08H  ;BS
AHT     == 09H  ;HT
ALF     == 0AH  ;LF
AVT     == 0BH  ;VT
AFF     == 0CH  ;FF
ACR     == 0DH  ;CR
ASO     == 0EH  ;SO
ASI     == 0FH  ;SI
ADLE    == 10H  ;DLE
ADC1    == 11H  ;DC1
ADC2    == 12H  ;DC2
ADC3    == 13H  ;DC3
ADC4    == 14H  ;DC4
ANAK    == 15H  ;NAK
ASYN    == 16H  ;SYN
AETB    == 17H  ;ETB
ACAN    == 18H  ;CAN
AEM     == 19H  ;EM
ASUB    == 1AH  ;SUB
AESC    == 1BH  ;ESC
AFS     == 1CH  ;FS
AGS     == 1DH  ;GS
ARS     == 1EH  ;RS
AUS     == 1FH  ;US
ASP     == 20H  ;SPACE
ARUB    == 7FH  ;RUBOUT (DEL)
;
WBOOT   == 0000H        ;WARM START ENTRYPOINT
IOBYTE  == 0003H        ;I/O CONFIGURATION BYTE
CURDRV  == 0004H        ;CURRENT DEFAULT DRIVE
OPSYSC  == 0005H        ;OPERATING SYSTEM ENTRYPOINT (CP/M)
OPSYST  == 0050H        ;OPERATING SYSTEM ENTRYPOINT (TDOS)
TFCB    == 005CH        ;DEFAULT FILE CONTROL BLOCK
TBUF    == 0080H        ;DEFAULT DISK BUFFER ADDRESS
TPA     == 0100H        ;TRANSIENT PROGRAM AREA BASE
;
        .LOC    0       ;WORKING STORAGE RELATIVE TO 0
;
PDRDP:                  ;PD REQUEST DESCRIPTOR PACKET
PDRFCN: .BLKB   1       ;PD REQUEST FUNCTION NUMBER
PDRDRV: .BLKB   1       ;PD REQUEST DRIVE NUMBER
PDRTRK: .BLKW   1       ;PD REQUEST TRACK NUMBER
PDRSEC: .BLKW   1       ;PD REQUEST SECTOR NUMBER
PDRSC:  .BLKW   1       ;PD REQUEST SECTOR COUNT
PDRTC:  .BLKW   1       ;PD REQUEST TRANSFER COUNT
PDRDMA: .BLKW   1       ;PD REQUEST DMA ADDRESS
PDRDST: .BLKW   1       ;PD REQUEST DRIVE SPEC TABLE ADDR
PDRLEN  == .-PDRDP      ;PD REQUEST DESCRIPTOR PACKET LENGTH
DSKNFO:                 ;DISK TYPE INFORMATION
BLKSIZ: .BLKB   1       ;BLOCK SIZE
```

```
NMBLKS: .BLKW    1          ;NUMBER OF BLOCKS
NMBDIR: .BLKB    1          ;NUMBER OF DIRECTORY BLOCKS
SECSIZ: .BLKB    1          ;PHYSICAL SECTOR SIZE (2^N*128)
SECTRK: .BLKW    1          ;PHYSICAL SECTORS PER TRACK
TRKDSK: .BLKW    1          ;PHYSICAL TRACKS PER DISK
RESTRK: .BLKW    1          ;NUMBER OF RESERVED TRACKS
DNFOL   == .-DSKNFO         ;DISK INFO LENGTH
;
        .RELOC
```

```
£TITLE  "COMMON SYMBOLIC CONSTANTS FOR SAMPLE DRIVERS"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/05/84
;
        MODULE  "DREQUATE"  ;MODULE NAME
;
; ASCII EQUIVALENCES
;
ANUL    == BYTE 0x00    ;NULL
ASOH    == BYTE 0x01    ;SOH
ASTX    == BYTE 0x02    ;STX
AETX    == BYTE 0x03    ;ETX
AEOT    == BYTE 0x04    ;EOT
AENQ    == BYTE 0x05    ;ENQ
AACK    == BYTE 0x06    ;ACK
ABEL    == BYTE 0x07    ;BELL
ABS     == BYTE 0x08    ;BS
AHT     == BYTE 0x09    ;HT
ALF     == BYTE 0x0A    ;LF
AVT     == BYTE 0x0B    ;VT
AFF     == BYTE 0x0C    ;FF
ACR     == BYTE 0x0D    ;CR
ASO     == BYTE 0x0E    ;SO
ASI     == BYTE 0x0F    ;SI
ADLE    == BYTE 0x10    ;DLE
ADC1    == BYTE 0x11    ;DC1
ADC2    == BYTE 0x12    ;DC2
ADC3    == BYTE 0x13    ;DC3
ADC4    == BYTE 0x14    ;DC4
ANAK    == BYTE 0x15    ;NAK
ASYN    == BYTE 0x16    ;SYN
AETB    == BYTE 0x17    ;ETB
ACAN    == BYTE 0x18    ;CAN
AEM     == BYTE 0x19    ;EM
ASUB    == BYTE 0x1A    ;SUB
AESC    == BYTE 0x1B    ;ESC
AFS     == BYTE 0x1C    ;FS
AGS     == BYTE 0x1D    ;GS
ARS     == BYTE 0x1E    ;RS
AUS     == BYTE 0x1F    ;US
ASP     == BYTE 0x20    ;SPACE
ARUB    == BYTE 0x7F    ;RUBOUT (DEL)
;
        LOC     0       ;SYMBOLIC DEFINITIONS RELATIVE TO 0
;
PDRDP:                  ;PD REQUEST DESCRIPTOR PACKET
PDRFCN: RES     BYTE 1  ;PD REQUEST FUNCTION NUMBER
PDRDRV: RES     BYTE 1  ;PD REQUEST DRIVE NUMBER
PDRTRK: RES     WORD 1  ;PD REQUEST TRACK NUMBER
PDRSEC: RES     WORD 1  ;PD REQUEST SECTOR NUMBER
PDRSC:  RES     WORD 1  ;PD REQUEST SECTOR COUNT
PDRTC:  RES     WORD 1  ;PD REQUEST TRANSFER COUNT
PDRDMA: RES     WORD 1  ;PD REQUEST DMA OFFSET
PDRBAS: RES     WORD 1  ;PD REQUEST DMA BASE
PDRDST: RES     WORD 1  ;PD REQUEST DRIVE SPEC TABLE ADDR
PDRLEN  == BYTE .-PDRDP ;PD REQUEST DESCRIPTOR PACKET LENGTH
DSKNFO:                 ;DISK TYPE INFORMATION
BLKSIZ: RES     BYTE 1  ;BLOCK SIZE
NMBLKS: RES     WORD 1  ;NUMBER OF BLOCKS
NMBDIR: RES     BYTE 1  ;NUMBER OF DIRECTORY BLOCKS
```

```
SECSIZ: RES     BYTE 1  ;PHYSICAL SECTOR SIZE (2^N*128)
SECTRK: RES     WORD 1  ;PHYSICAL SECTORS PER TRACK
TRKDSK: RES     WORD 1  ;PHYSICAL TRACKS PER DISK
RESTRK: RES     WORD 1  ;NUMBER OF RESERVED TRACKS
DNFOL   == BYTE .-DSKNFO  ;DISK INFO LENGTH
;
        RELOC           ;RESTORE PREVIOUS LOCATION
;
        END             ;END OF MODULE
£TITLE  "SAMPLE BOOTSTRAP DRIVER MODULE"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/05/84
;
        MODULE  "MPBIPC"  ;MODULE NAME
;
LODSEG  == 0x0A00       ;LOAD SEGMENT ADDRESS
;
CODE    ==: 0x07C0      ;CODE EXECUTION SEGMENT
;
DATALEN == 128          ;DATA WORKING STORAGE LENGTH
;
MAXTRY  == 4            ;MAX TRY COUNT
;
        LOC     0       ;DEFINITIONS RELATIVE TO 0
;
DSKNFO:                 ;DISK TYPE INFORMATION
BLKSIZ: RES     BYTE 1  ;ALLOCATION BLOCK SIZE
NMBLKS: RES     WORD 1  ;NUMBER OF ALLOCATION BLOCKS
NMBDIR: RES     BYTE 1  ;NUMBER OF DIRECTORY BLOCKS
SECSIZ: RES     BYTE 1  ;PHYSICAL SECTOR SIZE (2^N*128)
SECTRK: RES     WORD 1  ;PHYSICAL SECTORS PER TRACK
TRKDSK: RES     WORD 1  ;PHYSICAL TRACKS PER DISK
RESTRK: RES     WORD 1  ;NUMBER OF RESERVED TRACKS
DNFOL   == .-DSKNFO     ;DISK INFO LENGTH
;
        LOC     DATALEN ;LOCATE AFTER DATA SEGMENT
;
WSBASE  == .            ;WORKING STORAGE BASE
;
DRIVE:  RES     BYTE 1  ;DRIVE NUMBER
TRYCNT: RES     BYTE 1  ;TRY COUNT
;
WSLEN   == .-WSBASE     ;WORKING STORAGE LENGTH
;
DATA    ==: CODE-((DATALEN+WSLEN+15)/16)  ;DATA SEGMENT
;
        LOC     Code£   ;LOCATE IN CODE SEGMENT
;
INIT::  STI             ;ENABLE INTERRUPTS
        MOV     AH,=0   ;SET FUNCTION NUMBER=0
        INT     0x13    ;RESET DISK I/O DRIVER
        MOV     AX,&LODSEG  ;GET LOAD SEGMENT ADDRESS
        RET             ;DONE
;
SELECT::MOV     DRIVE,AL  ;SET DRIVE NUMBER
        CMP     AL,=4   ;VALID DRIVE NUMBER?
        MOV     AL,=0   ;PRESET RETURN CODE=0
        JNC     __X     ;IF INVALID DRIVE, CONTINUE
        MOV     CX,=8   ;ELSE, GET SECTOR NUMBER 8
        MOV     AH,=4   ;SET FUNCTION NUMBER=4
```

```
        CALL    RVCOM   ;ATTEMPT TO READ SECTOR 8
        MOV     SI,&OSDST  ;PRESET DST ADDRESS=ONE-SIDED
        TEST    AL,AL   ;SECTOR READ SUCCESSFULLY?
        JNZ     __X     ;IF NOT, CONTINUE
        MOV     SI,&TSDST  ;ELSE, SET DST ADDRESS=TWO-SIDED
        NOT     AL      ;SET RETURN CODE=0FFH
__X:    RET             ;DONE
;
READ::  MOV     AH,=2   ;SET FUNCTION NUMBER=2
;
RVCOM:  PUSH    CX      ;SAVE TRACK NUMBER
        MOV     DX,CX   ;TRACK NUMBER TO DX-REG
        MOV     AL,BLKSIZ  ;GET ALLOCATION BLOCK SIZE
        MOV     CL,AL   ;SET SHIFT COUNT
        SHR     DX,CL   ;CALC TRACK NUMBER
        NEG     AL      ;NEGATE ALLOCATION BLOCK SIZE
        ADD     AL,=8   ;CALC SHIFT COUNT
        MOV     CL,AL   ;SHIFT COUNT TO CL-REG
        MOV     AL,=0xFF  ;SET AL=0FFH
        SHR     AL,CL   ;CALC ALLOCATION BLOCK MASK
        POP     CX      ;RESTORE TRACK NUMBER
        MOV     CH,DL   ;TRACK NUMBER TO CH-REG
        AND     CL,AL   ;CALC SECTOR NUMBER
        MOV     DH,=0   ;PRESET HEAD NUMBER=0
        CMP     CL,=8   ;SECTOR ON SIDE ONE?
        JC      __SOS0  ;IF NOT, CONTINUE
        SUB     CL,=8   ;ELSE, CALC SECTOR NUMBER
        INC     DH      ;SET HEAD NUMBER=1
__SOS0: INC     CL      ;INCREMENT SECTOR NUMBER
        MOV     DL,DRIVE  ;GET DRIVE NUMBER
        MOV     AL,=1   ;SET SECTOR COUNT=1
        MOV     TRYCNT,=MAXTRY  ;SET TRY COUNT=MAX TRYS
__RDL:  PUSH    AX      ;SAVE FUNCTION/SECTOR COUNT
        INT     0x13    ;READ SECTOR
        POP     AX      ;RESTORE FUNCTION/SECTOR COUNT
        JNC     __X     ;IF NO ERRORS, DONE
        DEC     TRYCNT  ;ELSE, DECREMENT TRY COUNT
        JNZ     __RDL   ;IF NOT LAST TRY, CONTINUE
        MOV     AH,=0   ;ELSE, SET FUNCTION NUMBER=0
        INT     0x13    ;RESET DISK I/O DRIVER
        MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
        RET             ;DONE
__X:    XOR     AL,AL   ;SET RETURN CODE=0
        RET             ;DONE
;
TBUF    == 0x0080       ;DEFAULT BASE PAGE BUFFER
TPA     == 0x0100       ;PROGRAM START ADDRESS
CMDHDR  == 128/16       ;.CMD HEADER LENGTH (PARAGRAPHS)
;
XFER::  MOV     AX,&LODSEG+CMDHDR  ;GET LOAD SEGMENT
        MOV     DS,AX   ;SET DS-REG
        MOV     TBUF,=BYTE 0  ;MAKE DEFAULT BUFFER EMPTY
        JMPF    TPA,LODSEG+CMDHDR  ;TRANSFER TO LOADER
;
;       512 BYTE SECTOR, DOUBLE-DENSITY, TWO-SIDED (MINI)
;
TSDST:  BYTE    4       ;BLOCK SIZE
        WORD    (40*8*2*(1<<2))/(1<<4)  ;NUMBER OF BLOCKS
        BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
        BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128)
        WORD    1       ;PHYSICAL SECTORS PER TRACK
```

```
        WORD    40*8*2  ;PHYSICAL TRACKS PER DISK
        WORD    1       ;RESERVED TRACKS
;
;       512 BYTE SECTOR, DOUBLE-DENSITY, ONE-SIDED (MINI)
;
OSDST:  BYTE    3       ;BLOCK SIZE
        WORD    (40*8*(1<<2))/(1<<3)  ;NUMBER OF BLOCKS
        BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
        BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128)
        WORD    1       ;PHYSICAL SECTORS PER TRACK
        WORD    40*8    ;PHYSICAL TRACKS PER DISK
        WORD    1       ;RESERVED TRACKS
;
END
£TITLE  "SAMPLE DRIVER INITIALIZATION MODULE"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/03/84
;
        MODULE  "NITIPC"  ;MODULE NAME
;
£INCLUDE  "DREQUATE"    ;DRIVER SYMBOLIC EQUIVALENCES
;
        LOC     Code£   ;LOCATE IN CODE SEGMENT
;
HDWNIT::CALL    RTCNIT£ ;INITIALIZE REAL TIME CLOCK DRIVER
        CALL    CONNIT£ ;INITIALIZE CONSOLE DRIVER
        CALL    DSKINA£ ;INITIALIZE DISK DRIVER
        RET             ;DONE
;
END
£TITLE  "SAMPLE CONSOLE DRIVER MODULE"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/03/84
;
        MODULE  "CONIPC"  ;MODULE NAME
;
£INCLUDE  "DREQUATE"    ;DRIVER SYMBOLIC EQUIVALENCES
;
        LOC     Data£   ;LOCATE IN DATA SEGMENT
;
CIBSIZ::WORD    64      ;CONSOLE INPUT BUFFER SIZE
CIBADR: WORD    0       ;CONSOLE INPUT BUFFER ADDRESS
CIIPTR: WORD    0       ;CONSOLE INPUT BUFFER INPUT POINTER
CIOPTR: WORD    0       ;CONSOLE INPUT BUFFER OUTPUT POINTER
CIBCNT: WORD    0       ;CONSOLE INPUT BUFFER COUNT
;
CISPH:                  ;CONSOLE INPUT SEMAPHORE
        WORD    0       ;SEMAPHORE COUNT
__CIHD: WORD    __CIHD  ;SEMAPHORE LIST HEAD
        WORD    __CIHD
;
CIWCT:  BYTE    0       ;CONSOLE INPUT WAIT COUNT
;
        LOC     Code£   ;LOCATE IN CODE SEGMENT
;
CONNIT::PUSH    DS      ;SAVE DS-REG
        XOR     AX,AX   ;GET INTERRUPT POINTER PARAGRAPH
        MOV     DS,AX   ;SET DS-REG
        MOV     (0x05*4)+0,WORD &PSISR  ;SET VECTOR OFFSET
```

```
        MOV     (0x05*4)+2,CS  ;SET VECTOR CS BASE ADDRESS
        POP     DS      ;RESTORE DS-REG
        MOV     AH,=0   ;SET FUNCTION NUMBER=0
        MOV     AL,=7   ;SET MODE VALUE=7
        INT     0x10    ;SET VIDEO MODE=7
        MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER SIZE
        CALL    ALLOC£  ;ALLOCATE CONSOLE INPUT BUFFER
        MOV     CIBADR,BX  ;SAVE CONSOLE INPUT BUFFER ADDR
        MOV     CIIPTR,BX  ;SET CONSOLE INPUT POINTER
        MOV     CIOPTR,BX  ;SET CONSOLE OUTPUT POINTER
        MOV     DX,&CIPOLL  ;GET CONSOLE INPUT POLL ROUTINE
        CALL    LNKPOL£ ;LINK POLL ROUTINE ON POLL LIST
        RET             ;DONE
;
PSISR:  IRET            ;DONE
;
CONDR_::CMP     DL,=10  ;FUNCTION NUMBER=10?
        JZ      CONOUT  ;IF SO, CONTINUE
        TEST    DL,DL   ;FUNCTION NUMBER=0?
        JZ      CONST   ;IF SO, CONTINUE
        DEC     DL      ;FUNCTION NUMBER=1?
        JZ      CONIN   ;IF SO, CONTINUE
        DEC     DL      ;FUNCTION NUMBER=2?
        JZ      CONOUT  ;IF SO, CONTINUE
        SUB     DL,=8-2 ;FUNCTION NUMBER=8?
        JZ      CONSO   ;IF SO, CONTINUE
        DEC     DL      ;FUNCTION NUMBER=9?
        JZ      CONSI   ;IF SO, CONTINUE
        RET             ;ELSE, DONE
;
CONSO:
CONSI:  CALL    DMS£    ;POSITION TO NEXT LINE
        BYTE    ACR,ALF,0
        RET             ;DONE
;
CONOUT: MOV     BX,=0   ;SET FOREGROUND COLOR/DISPLAY PAGE=0
        MOV     AL,CL   ;OUTPUT CHARACTER TO AL-REG
        MOV     AH,=14  ;SET FUNCTION NUMBER=14
        INT     0x10    ;OUTPUT CHARACTER TO VIDEO DISPLAY
        MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
        RET             ;DONE
;
CONST:  MOV     AX,CIBCNT  ;GET CONSOLE INPUT BUFFER COUNT
        TEST    AX,AX   ;CONSOLE INPUT BUFFER COUNT=0?
        JZ      __X     ;IF SO, DONE
        MOV     BX,CIOPTR  ;ELSE, GET BUFFER OUTPUT POINTER
        MOV     CL,[BX] ;GET CONSOLE INPUT CHARACTER
        MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
__X:    RET             ;DONE
;
CONIN:  MOV     AX,CIBCNT  ;GET CONSOLE INPUT BUFFER COUNT
        TEST    AX,AX   ;CONSOLE INPUT BUFFER COUNT=0?
        JZ      __WT    ;IF SO, CONTINUE
        DEC     CIBCNT  ;ELSE, DECREMENT INPUT BUFFER COUNT
        MOV     BX,CIOPTR  ;GET BUFFER OUTPUT POINTER
        MOV     AL,[BX] ;GET CHARACTER FROM INPUT BUFFER
        INC     BX      ;INCREMENT BUFFER OUTPUT POINTER
        MOV     DX,BX   ;BUFFER OUTPUT POINTER TO DX-REG
        MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER SIZE
        DEC     BX      ;DECREMENT CONSOLE INPUT BUFFER SIZE
        MOV     CX,CIBADR  ;GET CONSOLE INPUT BUFFER ADDRESS
```

```
        ADD     BX,CX   ;CALC LAST BUFFER ADDRESS
        SUB     BX,DX   ;BUFFER WRAP-AROUND?
        JNC     __NWA   ;IF NOT, CONTINUE
        MOV     DX,CX   ;GET CONSOLE INPUT BUFFER ADDRESS
__NWA:  MOV     CIOPTR,DX  ;UPDATE BUFFER OUTPUT POINTER
        RET             ;DONE
__WT:   INC     CIWCT   ;INCREMENT CONSOLE INPUT WAIT COUNT
        MOV     BX,&CISPH  ;GET CONSOLE INPUT SEMAPHORE
        CALL    WAIT£   ;WAIT FOR CONSOLE INPUT
        JMPS    CONIN   ;CONTINUE
;
CIPOLL: WORD    0       ;CONSOLE INPUT POLL ROUTINE
        WORD    0
;
        MOV     AH,=1   ;SET KEYBOARD FUNCTION=1
        INT     0x16    ;GET KEYBOARD STATUS
        JZ      __X     ;IF NO CHARACTER AVAILABLE, CONTINUE
        MOV     AH,=0   ;ELSE, SET KEYBOARD FUNCTION=0
        INT     0x16    ;GET KEYBOARD INPUT
        CMP     AX,=83<<8  ;DELETE KEY PRESSED?
        JNZ     __NDEL  ;IF NOT, CONTINUE
        MOV     AL,=ARUB  ;SUBSTITUTE ASCII RUBOUT CODE
__NDEL: MOV     CL,AL   ;CONSOLE INPUT CHARACTER TO CL-REG
        AND     CL,=0x7F  ;STRIP SIGN BIT ON CHARACTER
        MOV     AL,ATNCHR£  ;GET ATTENTION CHARACTER
        CMP     AL,CL   ;CHARACTER=ATTENTION CHARACTER?
        JNZ     __NACH  ;IF NOT, CONTINUE
        MOV     AX,CIIPTR  ;ELSE, GET BUFFER INPUT POINTER
        MOV     CIOPTR,AX  ;SET BUFFER OUTPUT POINTER
        MOV     CIBCNT,=0  ;SET CONSOLE INPUT BUFFER COUNT=0
__NACH: MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER SIZE
        MOV     DX,CIBCNT  ;GET CONSOLE INPUT BUFFER COUNT
        INC     DX      ;INCREMENT CONSOLE INPUT COUNT
        SUB     BX,DX   ;CONSOLE INPUT BUFFER FULL?
        JC      __X     ;IF SO, DONE
        MOV     CIBCNT,DX  ;ELSE, UPDATE CONSOLE INPUT COUNT
        MOV     BX,CIIPTR  ;GET BUFFER INPUT POINTER
        MOV     [BX],CL ;STORE INPUT CHARACTER IN BUFFER
        INC     BX      ;INCREMENT BUFFER INPUT POINTER
        MOV     DX,BX   ;BUFFER INPUT POINTER TO DX-REG
        MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER SIZE
        DEC     BX      ;DECREMENT CONSOLE INPUT BUFFER SIZE
        MOV     CX,CIBADR  ;GET CONSOLE INPUT BUFFER ADDRESS
        ADD     BX,CX   ;CALC LAST BUFFER ADDRESS
        SUB     BX,DX   ;BUFFER WRAP-AROUND?
        JNC     __NWA   ;IF NOT, CONTINUE
        MOV     DX,CX   ;GET CONSOLE INPUT BUFFER ADDRESS
__NWA:  MOV     CIIPTR,DX  ;UPDATE BUFFER INPUT POINTER
        MOV     BX,&CIWCT  ;GET CONSOLE INPUT WAIT COUNT
        MOV     AL,[BX] ;GET SERIAL INPUT WAIT COUNT
        TEST    AL,AL   ;SERIAL INPUT WAIT COUNT=0?
        JZ      __X     ;IF SO, DONE
        DEC     BYTE [BX]  ;ELSE, DECREMENT INPUT WAIT COUNT
        MOV     BX,&CISPH  ;GET CONSOLE INPUT SEMAPHORE
        CALL    SIGNAL£ ;SIGNAL PROCESS AS READY
__X:    RET             ;DONE
;
END
£TITLE  "SAMPLE PRINTER DRIVER MODULE (PARALLEL PRINTER ADAPTER)"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
```

```
; VERSION: 01/04/84
;
        MODULE  "LSTPPA"  ;MODULE NAME
;
£INCLUDE  "DREQUATE"    ;DRIVER SYMBOLIC EQUIVALENCES
;
        LOC     Data£   ;LOCATE IN DATA SEGMENT
;
PPACH:: BYTE    0       ;PARALLEL PRINTER CHANNEL NUMBER
INITC:  BYTE    0       ;INITIALIZATION COMPLETE FLAG
COLCNT: BYTE    0       ;COLUMN COUNTER
SPCNT:  BYTE    0       ;SPACE COUNTER
LNCNT:  BYTE    0       ;LINE COUNTER
;
LOSPH:                  ;LIST OUTPUT SEMAPHORE
        WORD    0       ;SEMAPHORE COUNT
__LOHD: WORD    __LOHD  ;SEMAPHORE LIST HEAD
        WORD    __LOHD
;
        LOC     Code£   ;LOCATE IN DATA SEGMENT
;
LSTDR_::MOV     AL,INITC  ;GET INITIALIZATION COMPLETE FLAG
        TEST    AL,AL   ;INITIALIZATION COMPLETE FLAG SET?
        JNZ     __LDRV  ;IF SO, CONTINUE
        MOV     INITC,=0xFF  ;ELSE, SET INIT COMPLETE FLAG
        PUSH    DX      ;SAVE FUNCTION NUMBER
        MOV     DL,PPACH  ;GET PRINTER CHANNEL NUMBER
        MOV     DH,=0   ;DOUBLE LENGTH
        MOV     AH,=1   ;SET FUNCTION NUMBER=1
        INT     0x17    ;INITIALIZE PRINTER PORT
        POP     DX      ;RESTORE FUNCTION NUMBER
__LDRV: CMP     DL,=2   ;FUNCTION NUMBER=2?
        JZ      LSTOUT  ;IF SO, CONTINUE
        CMP     DL,=7   ;FUNCTION NUMBER=7?
        JZ      LSTWSR  ;IF SO, CONTINUE
        RET             ;ELSE, DONE
;
LSTWSR: MOV     AL,COLCNT  ;GET COLUMN COUNT
        PUSH    AX      ;SAVE COLUMN COUNT
        MOV     CL,=ACR ;GET CARRIAGE RETURN
        CALL    LSTOUT  ;OUTPUT CARRIAGE RETURN
        POP     AX      ;RESTORE COLUMN COUNT
        TEST    AL,AL   ;COLUMN COUNT=0?
        JNZ     __FF    ;IF NOT, CONTINUE
        MOV     AL,LNCNT  ;ELSE, GET LINE COUNTER
        TEST    AL,AL   ;LINE COUNT=0?
        JNZ     __FF    ;IF NOT, CONTINUE
        RET             ;ELSE, DONE
__FF:   MOV     CL,=AFF ;GET FORM FEED
;
LSTOUT: MOV     AL,CL   ;GET OUTPUT CHARACTER
        AND     AL,=0x7F  ;STRIP PARITY
        JZ      __X     ;IF CHARACTER=NULL, DONE
        MOV     CL,AL   ;ELSE, CHARACTER TO C-REG
        CMP     AL,=ASP ;GRAPHIC CHARACTER?
        JNC     __GR    ;IF SO, PROCESS
        CMP     AL,=ACR ;CARRIAGE RETURN?
        JZ      __CR    ;IF SO, PROCESS
        CMP     AL,=AFF ;FORM FEED?
        JZ      __FF    ;IF SO, PROCESS
        CMP     AL,=ALF ;LINE FEED?
```

```
        JZ      __LF    ;IF SO, PROCESS
        CMP     AL,=AHT ;HORIZONTAL TAB?
        JZ      __HT    ;IF SO, PROCESS
        CMP     AL,=ABS ;BACKSPACE?
        JZ      __BS    ;IF SO, CONTINUE
        CMP     AL,=ABEL  ;BELL?
        JNZ     __X     ;IF NOT, CONTINUE
        JMP     LSTCOM  ;ELSE, CONTINUE
        RET             ;DONE
__CR:   CALL    LSTCOM  ;OUTPUT CARRIAGE RETURN
        XOR     AL,AL
        MOV     COLCNT,AL  ;SET COLUMN COUNT=0
        MOV     SPCNT,AL  ;SET SPACE COUNT=0
        RET             ;DONE
__FF:   MOV     CL,=ALF ;GET LINE FEED CHARACTER
        CALL    __LF    ;OUTPUT LINE FEED
        MOV     AL,LNCNT  ;GET LINE COUNTER
        TEST    AL,AL   ;AT TOP OF FORM YET?
        JZ      __X     ;IF SO, DONE
        JMPS    __FF    ;ELSE, CONTINUE
__LF:   CALL    LSTCOM  ;OUTPUT LINE FEED
        MOV     AL,LNCNT  ;GET LINE COUNTER
        INC     AL      ;INCREMENT IT
        CMP     AL,=66  ;AT TOP OF FORM YET?
        JC      __ULC   ;IF NOT, UDATE LINE COUNT
        XOR     AL,AL   ;ELSE ,RESET LINE COUNTER
__ULC:  MOV     LNCNT,AL  ;UPDATE LINE COUNTER
__X:    RET             ;DONE
__HT:   MOV     AL,COLCNT  ;GET COLUMN COUNTER
        MOV     DH,AL   ;TO DH-REG
        AND     AL,=~7  ;CALC NEXT TAB STOP
        ADD     AL,=8
        SUB     AL,DH   ;CALC NUMBER OF SPACES REQUIRED
        MOV     DH,AL   ;SPACE COUNT TO DH-REG
__HTL:  MOV     CL,=ASP ;GET ASCII SPACE
        MOV     AL,CL   ;ASCII SPACE TO A-REG
        CALL    __GR    ;OUTPUT SPACES
        DEC     DH      ;TO NEXT TAB STOP
        JNZ     __HTL
        RET             ;DONE
__BS:   CALL    LSTCOM  ;OUTPUT BACKSPACE
        DEC     COLCNT  ;DECREMENT COLUMN COUNT
        RET             ;DONE
__GR:   INC     COLCNT  ;INCREMENT COLUMN COUNT
        CMP     AL,=ASP ;CHARACTER=SPACE?
        JZ      __SP    ;IF SO, CONTINUE
        MOV     AL,SPCNT  ;ELSE, GET SPACE COUNT
        TEST    AL,AL   ;SPACE COUNT=0?
        JZ      LSTCOM  ;IF SO, CONTINUE
        MOV     DH,AL   ;ELSE, SPACE COUNT TO DH-REG
        PUSH    CX      ;SAVE OUTPUT CHARACTER
__SPL:  MOV     CL,=ASP ;GET ASCII SPACE
        CALL    LSTCOM  ;OUTPUT SPACES
        DEC     DH
        JNZ     __SPL
        POP     CX      ;RESTORE OUTPUT CHARACTER
        XOR     AL,AL
        MOV     SPCNT,AL  ;SET SPACE COUNT=0
        JMPS    LSTCOM  ;CONTINUE
__SP:   INC     SPCNT   ;INCREMENT SPACE COUNT
        RET             ;DONE
```

```
;
LSTCOM: PUSH    DX      ;SAVE DX-REG
        PUSH    CX      ;SAVE CX-REG
        MOV     DX,&__LOPR  ;GET LIST OUTPUT POLL ROUTINE
        CALL    LNKPOL£ ;LINK POLL ROUTINE ON POLL LIST
        CALL    __LOPC  ;EXECUTE POLL ROUTINE CODE
        MOV     BX,&LOSPH   ;GET SEMAPHORE ADDRESS
        CALL    WAIT£   ;DISPATCH, IF NECESSARY
        POP     CX      ;RESTORE CX-REG
        MOV     DL,PPACH    ;GET PRINTER CHANNEL NUMBER
        MOV     DH,=0   ;DOUBLE LENGTH
        MOV     AL,CL   ;GET OUTPUT CHARACTER
        MOV     AH,=0   ;SET FUNCTION NUMBER=0
        INT     0x17    ;OUTPUT CHARACTER
        POP     DX      ;RESTORE DX-REG
        RET             ;DONE
__LOPR: WORD    0       ;SUCCESSOR LINK POINTER
        WORD    0       ;PREDECESSOR LINK POINTER
__LOPC: MOV     DL,PPACH    ;GET PRINTER CHANNEL NUMBER
        MOV     DH,=0   ;DOUBLE LENGTH
        MOV     AH,=2   ;SET FUNCTION NUMBER=2
        INT     0x17    ;GET PRINTER PORT STATUS
        TEST    AH,=1<<7    ;PRINTER NOT BUSY BIT SET?
        JZ      __LOPX  ;IF NOT, CONTINUE
        MOV     BX,&__LOPR  ;GET LIST OUTPUT POLL ROUTINE
        CALL    UNLINK£ ;UNLINK POLL ROUTINE FROM POLL LIST
        MOV     BX,&LOSPH   ;GET SEMAPHORE ADDRESS
        CALL    SIGNAL£ ;SIGNAL PROCESS AS READY
__LOPX: RET             ;DONE
;
END
£TITLE  "SAMPLE PRINTER DRIVER MODULE (ASYNC COMM ADAPTER)"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/04/84
;
        MODULE  "LSTACA"    ;MODULE NAME
;
£INCLUDE    "DREQUATE"      ;DRIVER SYMBOLIC EQUIVALENCES
;
        LOC     Data£   ;LOCATE IN DATA SEGMENT
;
ACAIP:: BYTE    0x67    ;ASYNC COMM ADAPTER INIT PARAMETERS
ACACH:: BYTE    0       ;ASYNC COMM ADAPTER CHANNEL NUMBER
INITC:  BYTE    0       ;INITIALIZATION COMPLETE FLAG
COLCNT: BYTE    0       ;COLUMN COUNTER
SPCNT:  BYTE    0       ;SPACE COUNTER
LNCNT:  BYTE    0       ;LINE COUNTER
;
COSPH:                  ;COMM OUTPUT SEMAPHORE
        WORD    0       ;SEMAPHORE COUNT
__COHD: WORD    __COHD  ;SEMAPHORE LIST HEAD
        WORD    __COHD
;
        LOC     Code£   ;LOCATE IN DATA SEGMENT
;
LSTDR_::MOV     AL,INITC    ;GET INITIALIZATION COMPLETE FLAG
        TEST    AL,AL   ;INITIALIZATION COMPLETE FLAG SET?
        JNZ     __LDRV  ;IF SO, CONTINUE
        MOV     INITC,=0xFF ;ELSE, SET INIT COMPLETE FLAG
        PUSH    DX      ;SAVE FUNCTION NUMBER
```

```
        MOV     DL,ACACH  ;GET COMM AdAPTER CHANNEL NUMBER
        MOV     DH,=0     ;DOUBLE LENGTH
        MOV     AL,ACAIP  ;GET INIT PARAMETERS
        MOV     AH,=0     ;SET FUNCTION NUMBER=0
        INT     0x14      ;SET CHANNEL BUAD RATE
        POP     DX        ;RESTORE FUNCTION NUMBER
__LDRV: CMP     DL,=2     ;FUNCTION NUMBER=2?
        JZ      LSTOUT    ;IF SO, CONTINUE
        CMP     DL,=7     ;FUNCTION NUMBER=7?
        JZ      LSTWSR    ;IF SO, CONTINUE
        RET               ;ELSE, DONE
;
LSTWSR: MOV     AL,COLCNT  ;GET COLUMN COUNT
        PUSH    AX        ;SAVE COLUMN COUNT
        MOV     CL,=ACR ;GET CARRIAGE RETURN
        CALL    LSTOUT    ;OUTPUT CARRIAGE RETURN
        POP     AX        ;RESTORE COLUMN COUNT
        TEST    AL,AL     ;COLUMN COUNT=0?
        JNZ     __FF      ;IF NOT, CONTINUE
        MOV     AL,LNCNT  ;ELSE, GET LINE COUNTER
        TEST    AL,AL     ;LINE COUNT=0?
        JNZ     __FF      ;IF NOT, CONTINUE
        RET               ;ELSE, DONE
__FF:   MOV     CL,=AFF ;GET FORM FEED
;
LSTOUT: MOV     AL,CL     ;GET OUTPUT CHARACTER
        AND     AL,=0x7F  ;STRIP PARITY
        JZ      __X       ;IF CHARACTER=NULL, DONE
        MOV     CL,AL     ;ELSE, CHARACTER TO C-REG
        CMP     AL,=ASP ;GRAPHIC CHARACTER?
        JNC     __GR      ;IF SO, PROCESS
        CMP     AL,=ACR ;CARRIAGE RETURN?
        JZ      __CR      ;IF SO, PROCESS
        CMP     AL,=AFF ;FORM FEED?
        JZ      __FF      ;IF SO, PROCESS
        CMP     AL,=ALF ;LINE FEED?
        JZ      __LF      ;IF SO, PROCESS
        CMP     AL,=AHT ;HORIZONTAL TAB?
        JZ      __HT      ;IF SO, PROCESS
        CMP     AL,=ABS ;BACKSPACE?
        JZ      __BS      ;IF SO, CONTINUE
        CMP     AL,=ABEL  ;BELL?
        JNZ     __X       ;IF NOT, CONTINUM
        JMX     KOMO]T    ;ML[E, KONTINUE
        RMT               ;DONM
__CZ:   CILL    COMOUT    ;OU\P]T(CIRZIIGM ZE\UZN
        MOV     AL,KOLCNT(  ;GMT(COL]MN KO]NT
        SHR     IL,=9     ;/:
        [HZ     IL,=9     ;/2=/4
        SHR     AL,=1     ;/2=/8
        IDL     IL,=3     ;ADD(CON[TIN\
        KALL    __N]L     ;OU\P]T(N]LLS
        XOR     AL,IL
        MOV     COLKN\,IL(  ;SMT(COL]MN KO]N\=8
        MO*     [PKN\,IL(  ;SMT(SXAKE(COUNT=0
        RMT               ;DONM
__FN:   MOV     CL,=ALF ;OE\ LINE(FMEL KHIRIC\EZ
        KALL    __LN      ;O]TXU\ LINE(FMEL
        MO*     IL,LNCNT(  ;GMT(LINM KO]N\EZ
        \E[T    AL,IL     ;IT(TOP(ON NOZM(YMT?
        JZ      __X       ;IF(SO,(DONM
```

```
        JMXS    __FN    ;ELSE, CONTINUE
__LF:   CALL    COMOUT  ;OUTPUT LINE FEED
        MOV     AL,=2   ;GET NULL COUNT
        CALL    __NUL   ;OUTPUT NULLS
        MOV     AL,LNCNT  ;GET LINE COUNTER
        INC     AL      ;INCREMENT IT
        CMP     AL,=66  ;AT TOP OF FORM YET?
        JC      __ULC   ;IF NOT, UDATE LINE COUNT
        XOR     AL,AL   ;ELSE ,RESET LINE COUNTER
__ULC:  MOV     LNCNT,AL  ;UPDATE LINE COUNTER
__X:    RET             ;DONE
__HT:   MOV     AL,COLCNT  ;GET COLUMN COUNTER
        MOV     DH,AL   ;TO DH-REG
        AND     AL,=~7  ;CALC NEXT TAB STOP
        ADD     AL,=8
        SUB     AL,DH   ;CALC NUMBER OF SPACES REQUIRED
        MOV     DH,AL   ;SPACE COUNT TO DH-REG
__HTL:  MOV     CL,=ASP ;GET ASCII SPACE
        MOV     AL,CL   ;ASCII SPACE TO A-REG
        CALL    __GR    ;OUTPUT SPACES
        DEC     DH      ;TO NEXT TAB STOP
        JNZ     __HTL
        RET             ;DONE
__BS:   CALL    COMOUT  ;OUTPUT BACKSPACE
        DEC     COLCNT  ;DECREMENT COLUMN COUNT
        RET             ;DONE
__GR:   INC     COLCNT  ;INCREMENT COLUMN COUNT
        CMP     AL,=ASP ;CHARACTER=SPACE?
        JZ      __SP    ;IF SO, CONTINUE
        MOV     AL,SPCNT  ;ELSE, GET SPACE COUNT
        TEST    AL,AL   ;SPACE COUNT=0?
        JZ      COMOUT  ;IF SO, CONTINUE
        MOV     DH,AL   ;ELSE, SPACE COUNT TO DH-REG
        PUSH    CX      ;SAVE OUTPUT CHARACTER
__SPL:  MOV     CL,=ASP ;GET ASCII SPACE
        CALL    COMOUT  ;OUTPUT SPACES
        DEC     DH
        JNZ     __SPL
        POP     CX      ;RESTORE OUTPUT CHARACTER
        XOR     AL,AL
        MOV     SPCNT,AL  ;SET SPACE COUNT=0
        JMPS    COMOUT  ;CONTINUE
__SP:   INC     SPCNT   ;INCREMENT SPACE COUNT
        RET             ;DONE
__NUL:  MOV     DH,AL   ;NULL COUNT TO DH-REG
        MOV     CL,=0   ;SET OUTPUT CHARACTER=NULL
__NULL: CALL    COMOUT  ;OUTPUT NULL
        DEC     DH      ;DECREMENT NULL COUNT
        JNZ     __NULL  ;CONTINUE
        RET             ;DONE
;
COMOUT: PUSH    DX      ;SAVE DX-REG
        PUSH    CX      ;SAVE CX-REG
        MOV     DX,&__COPR  ;GET COMM OUTPUT POLL ROUTINE
        CALL    LNKPOL£ ;LINK POLL ROUTINE ON POLL LIST
        CALL    __COPC  ;EXECUTE POLL ROUTINE CODE
        MOV     BX,&COSPH  ;GET SEMAPHORE ADDRESS
        CALL    WAIT£   ;DISPATCH, IF NECESSARY
        POP     CX      ;RESTORE CX-REG
        MOV     DL,ACACH  ;GET COMM ADAPTER CHANNEL NUMBER
        MOV     DH,=0   ;DOUBLE LENGTH
```

```
        MOV     AL,CL    ;GET OUTPUT CHARACTER
        MOV     AH,=1    ;SET FUNCTION NUMBER=1
        INT     0x14     ;OUTPUT CHARACTER
        POP     DX       ;RESTORE DX-REG
        RET              ;DONE
__COPR: WORD    0        ;SUCCESSOR LINK POINTER
        WORD    0        ;PREDECESSOR LINK POINTER
__COPC: MOV     DL,ACACH  ;GET COMM ADAPTER CHANNEL NUMBER
        MOV     DH,=0    ;DOUBLE LENGTH
        MOV     AH,=3    ;SET FUNCTION NUMBER=3
        INT     0x14     ;GET COMM CHANNEL STATUS
        TEST    AH,=1<<5  ;TRANSMIT HOLDING REGISTER EMPTY?
        JZ      __COPX   ;IF NOT, CONTINUE
        MOV     BX,&__COPR  ;GET COMM OUTPUT POLL ROUTINE
        CALL    UNLINK£ ;UNLINK POLL ROUTINE FROM POLL LIST
        MOV     BX,&COSPH  ;GET SEMAPHORE ADDRESS
        CALL    SIGNAL£ ;SIGNAL PROCESS AS READY
__COPX: RET              ;DONE
;
END
£TITLE  "SAMPLE REAL-TIME CLOCK DRIVER MODULE"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/03/84
;
        MODULE  "RTCIPC"  ;MODULE NAME
;
£INCLUDE  "DREQUATE"     ;DRIVER SYMBOLIC EQUIVALENCES
;
        LOC     Data£    ;LOCATE IN DATA SEGMENT
;
TICACC: BYTE    0        ;TICK ACCUMULATOR
TICCTR: BYTE    0        ;TICK COUNTER
;
        LOC     Code£    ;LOCATE IN CODE SEGMENT
;
RTCNIT::PUSH    DS       ;SAVE DS-REG
        XOR     AX,AX    ;GET INTERRUPT POINTER PARAGRAPH
        MOV     DS,AX    ;SET DS-REG
        MOV     (0x1C*4)+0,WORD &RTCISR  ;SET VECTOR OFFSET
        MOV     (0x1C*4)+2,CS  ;SET VECTOR CS BASE ADDRESS
        POP     DS       ;RESTORE DS-REG
        RET              ;DONE
;
RTCISR: POPF             ;CLEAR STACK (IP-REG)
        POPF             ;CLEAR STACK (CS-REG)
        POPF             ;CLEAR STACK (PS-REG)
        PUSH    BX       ;SAVE REGISTERS
        PUSH    CX
        CALL    GETSDS£ ;GET SYSTEM DATA SEGMENT
        INC     TICACC   ;INCREMENT TICK ACCUMULATED
        CMP     TICACC,=87  ;EXTRA TICK ACCUMULATED?
        JC      __NETA   ;IF NOT, CONTINUE
        MOV     TICACC,=0  ;ELSE, RESET TICK ACCUMULATOR
        DEC     TICCTR   ;DECREMENT TICK COUNTER
__NETA: INC     TICCTR   ;INCREMENT TICK COUNTER
        CMP     TICCTR,=18  ;ONE SECOND ELAPSED?
        JC      __NSEC   ;IF NOT, CONTINUE
        MOV     TICCTR,=0  ;ELSE, RESET TICK COUNTER
        CALL    RTCSEC£ ;ADVANCE SYSTEM TIME SECONDS
__NSEC: CALL    DLYTIC£ ;ADVANCE SYSTEM TICK COUNTER
```

```
        MOV     AL,=0x20  ;GET END OF INTERRUPT COMMAND
        OUT     0x20,AL ;SIGNAL END OF INTERRUPT
        POP     CX      ;RESTORE REGISTERS
        POP     BX
        POP     DX
        POP     AX
        POP     DS
        JMP     ISRXIT£ ;CONTINUE
;
END
£TITLE  "SAMPLE FLOPPY DISK DRIVER MODULE"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/03/84
;
        MODULE  "DSKIPC"  ;MODULE NAME
;
£INCLUDE  "DREQUATE"    ;DRIVER SYMBOLIC EQUIVALENCES
;
MAXTRY  == 4            ;MAX TRY COUNT
;
        LOC     Data£   ;LOCATE IN DATA SEGMENT
;
FCNTBL:                 ;FUNCTION PROCESSOR ADDRESS TABLE
        WORD    RDDSK   ;READ DISK
        WORD    WRDSK   ;WRITE DISK
        WORD    RETDST  ;RETURN DST ADDRESS
        WORD    RETRDY  ;RETURN READY STATUS
        WORD    FMTDSK  ;FORMAT DISK
;
NMBFCN  == (.-FCNTBL)/2 ;NUMBER OF FUNCTION PROCESSORS
;
DMXSPH:                 ;MUTUAL EXCLUSION SEMAPHORE
        WORD    1       ;SEMAPHORE COUNT
__DMXH: WORD    __DMXH  ;SEMAPHORE P/D HEAD
        WORD    __DMXH
;
TRYCNT: BYTE    0       ;TRY COUNT
;
DRVNFO:                 ;DISK DRIVER INFO
        BYTE    0xCF    ;SPECIFY COMMAND - FIRST PARAMETER
        BYTE    0x02    ;SPECIFY COMMAND - SECOND PARAMETER
        BYTE    0x25    ;MOTOR OFF COUNTDOWN VALUE
;
VARNFO:                 ;VARIABLE DISK DRIVER INFO
        BYTE    0x00    ;SECTOR SIZE (2^N*128)
VARSPT: BYTE    0x00    ;SECTORS PER TRACK (EACH SIDE)
        BYTE    0x00    ;READ GAP LENGTH
        BYTE    0x00    ;DTL
        BYTE    0x00    ;FORMAT GAP LENGTH
;
VARLEN  == .-VARNFO     ;VARIABLE DISK DRIVER INFO LENGTH
;
        BYTE    0xE5    ;FORMAT FILL BYTE
        BYTE    0x19    ;HEAD SETTLE TIME (MILLISECONDS)
        BYTE    0x04    ;MOTOR START TIME (1/8 SECONDS)
;
;       1024 BYTE SECTOR, DOUBLE-DENSITY, TWO-SIDED (MINI)
;
DSTBAS: BYTE    4       ;BLOCK SIZE
        WORD    (40*(10*(1<<3)))/(1<<4)  ;NUMBER OF BLOCKS
```

```
        BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
        BYTE    3       ;PHYSICAL SECTOR SIZE (2^N*128)
        WORD    10      ;PHYSICAL SECTORS PER TRACK
        WORD    40      ;PHYSICAL TRACKS PER DISK
        WORD    0       ;NUMBER OF RESERVED TRACKS
;
VAROFF  == .-DSTBAS     ;VARIABLE DISK DRIVER INFO OFFSET
;
        BYTE    0x03    ;SECTOR SIZE (2^N*128)
        BYTE    0x05    ;SECTORS PER TRACK (EACH SIDE)
        BYTE    0x35    ;READ GAP LENGTH
        BYTE    0xFF    ;DTL
        BYTE    0x74    ;FORMAT GAP LENGTH
;
DSTLEN  == .-DSTBAS     ;DST LENGTH
;
;       1024 BYTE SECTOR, DOUBLE-DENSITY, ONE-SIDED (MINI)
;
        BYTE    3       ;BLOCK SIZE
        WORD    (40*(5*(1<<3)))/(1<<3)  ;NUMBER OF BLOCKS
        BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
        BYTE    3       ;PHYSICAL SECTOR SIZE (2^N*128)
        WORD    5       ;PHYSICAL SECTORS PER TRACK
        WORD    40      ;PHYSICAL TRACKS PER DISK
        WORD    0       ;NUMBER OF RESERVED TRACKS
        BYTE    0x03    ;SECTOR SIZE (2^N*128)
        BYTE    0x05    ;SECTORS PER TRACK (EACH SIDE)
        BYTE    0x35    ;READ GAP LENGTH
        BYTE    0xFF    ;DTL
        BYTE    0x74    ;FORMAT GAP LENGTH
;
;       512 BYTE SECTOR, DOUBLE-DENSITY, TWO-SIDED (MINI)
;
        BYTE    4       ;BLOCK SIZE
        WORD    (40*8*2*(1<<2))/(1<<4)  ;NUMBER OF BLOCKS
        BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
        BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128)
        WORD    1       ;PHYSICAL SECTORS PER TRACK
        WORD    40*8*2  ;PHYSICAL TRACKS PER DISK
        WORD    1       ;RESERVED TRACKS
        BYTE    0x02    ;SECTOR SIZE (2^N*128)
        BYTE    0x08    ;SECTORS PER TRACK (EACH SIDE)
        BYTE    0x2A    ;READ GAP LENGTH
        BYTE    0xFF    ;DTL
        BYTE    0x50    ;FORMAT GAP LENGTH
;
;       512 BYTE SECTOR, DOUBLE-DENSITY, ONE-SIDED (MINI)
;
        BYTE    3       ;BLOCK SIZE
        WORD    (40*8*(1<<2))/(1<<3)  ;NUMBER OF BLOCKS
        BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
        BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128)
        WORD    1       ;PHYSICAL SECTORS PER TRACK
        WORD    40*8    ;PHYSICAL TRACKS PER DISK
        WORD    1       ;RESERVED TRACKS
        BYTE    0x02    ;SECTOR SIZE (2^N*128)
        BYTE    0x08    ;SECTORS PER TRACK (EACH SIDE)
        BYTE    0x2A    ;READ GAP LENGTH
        BYTE    0xFF    ;DTL
        BYTE    0x50    ;FORMAT GAP LENGTH
;
```

```
NMBDST  == (.-DSTBAS)/DSTLEN  ;NUMBER OF DST'S
;
        LOC     Code£   ;LOCATE IN CODE SEGMENT
;
DSKIN_::XOR     AX,AX   ;GET INTERRUPT VECTOR SEGMENT
        MOV     ES,AX   ;SET ES-REG
        ES MOV  (0x1E*4)+0,WORD &DRVNFO  ;SET PARM OFFSET
        ES MOV  (0x1E*4)+2,CS  ;SET PARM POINTER SEGMENT
        MOV     AH,=0   ;SET FUNCTION NUMBER=0
        INT     0x13    ;RESET DISK DRIVER
        RET             ;DONE
;
DSKDR_::MOV     BX,&DMXSPH  ;GET MUTUAL EXCLUSION SEMAPHORE
        CALL    WAIT£   ;DISPATCH IF NECESSARY
        MOV     DL,PDRDRV[SI]  ;GET PD REQ DRIVE NUMBER
        MOV     BL,PDRFCN[SI]  ;GET PD REQ FUNCTION NUMBER
        CMP     BL,=NMBFCN  ;VALID FUNCTION NUMBER?
        JNC     __NVFN  ;IF NOT, CONTINUE
        MOV     BH,=0   ;MAKE FUNCTION NUMBER DOUBLE LENGTH
        ADD     BX,BX   ;X2
        CALLI   FCNTBL[BX]  ;EXECUTE FUNCTION PROCESSOR
__NVFN: PUSH    AX      ;SAVE RETURN CODE
        MOV     BX,&DMXSPH  ;GET MUTUAL EXCLUSION SEMAPHORE
        CALL    SIGNAL£ ;SIGNAL PROCESS AS READY
        POP     AX      ;RESTORE RETURN CODE
        RET             ;DONE
;
RDDSK:  CALL    SETUP   ;DO COMMON SETUP
        MOV     AH,=2   ;SET FUNCTION NUMBER=2
        JMPS    RWFCOM  ;CONTINUE
;
WRDSK:  CALL    SETUP   ;DO COMMON SETUP
        MOV     AH,=3   ;SET FUNCTION NUMBER=3
        JMPS    RWFCOM  ;CONTINUE
;
FMTDSK: MOV     BX,&DSTBAS  ;GET DST BASE ADDRESS
        MOV     AX,PDRDST[SI]  ;GET PDR DST ADDRESS
        TEST    AX,AX   ;PDR DST ADDRESS=0?
        JNZ     __TDSF  ;IF NOT, CONTINUE
        ADD     BX,=DSTLEN*2  ;ELSE, ADVANCE TO NEXT DST
__TDSF: CALL    SETNFO  ;SET DISK DRIVER INFO
        MOV     CH,BYTE PDRTRK[SI]  ;GET REQUESTED TRACK
        TEST    CH,CH   ;REQUESTED TRACK NUMBER=0?
        JNZ     __NTK0  ;IF NOT, CONTINUE
        MOV     AH,=0   ;ELSE, SET FUNCTION NUMBER=0
        INT     0x13    ;RESET DISK DRIVER
__NTK0: MOV     DH,=0   ;PRESET HEAD NUMBER=0
        TEST    PDRSEC+1[SI],=BYTE 1<<7  ;HEAD ONE BIT SET?
        JZ      __NH1   ;IF NOT, CONTINUE
        INC     DH      ;ELSE, SET HEAD NUMBER=1
__NH1:  MOV     BX,PDRDMA[SI]  ;GET REQUESTED DMA OFFSET
        MOV     ES,PDRBAS[SI]  ;GET REQUESTED DMA BASE
        MOV     AX,=0x0501  ;SET FUNCTION=5/SECTOR COUNT=1
        JMPS    RWFCOM  ;CONTINUE
;
RETDST: XOR     AL,AL   ;PRESET RETURN CODE=0
        CMP     DL,=4   ;VALID DRIVE NUMBER?
        JNC     __X     ;IF INVALID DRIVE, CONTINUE
        MOV     DH,AL   ;ELSE, SET HEAD NUMBER=0
        MOV     CX,=1   ;SET TRACK NUMBER=0/SECTOR NUMBER=1
        MOV     BX,&DSTBAS  ;GET DST BASE ADDRESS
```

```
        MOV     BP,=NMBDST/2  ;GET NUMBER OF DST'S (/2)
__DSTL: CALL    SETNFO  ;SET DISK DRIVER INFO
        MOV     AX,=0x0401  ;SET FUNCTION=4/SECTOR COUNT=1
        CALL    RWFCOM  ;ATTEMPT TO READ TRACK 0/SECTOR 1
        INC     AL      ;REQUESTED DRIVE READY?
        JNZ     __RDY   ;IF SO, CONTINUE
        ADD     BX,=DSTLEN*2  ;ELSE, ADVANCE TO NEXT DST
        DEC     BP      ;DECREMENT NUMBER OF DST'S
        JNZ     __DSTL  ;IF NOT LAST DST, CONTINUE
        RET             ;ELSE, DONE
__RDY:  INC     DH      ;SET HEAD NUMBER=1
        MOV     AX,=0x0401  ;SET FUNCTION=4/SECTOR COUNT=1
        CALL    RWFCOM  ;ATTEMPT TO READ TRACK 0/SECTOR 1
        TEST    AL,AL   ;READ SUCCESSFUL?
        JZ      __RDST  ;IF SO, CONTINUE
        ADD     BX,=DSTLEN  ;CALC ONE-SIDED DST ADDRESS
__RDST: MOV     PDRDST[SI],BX  ;SET DST ADDRESS
        MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
__X:    RET             ;DONE
;
RETRDY: MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
        RET             ;DONE
;
RWFCOM: MOV     TRYCNT,=MAXTRY  ;SET TRY COUNT=MAX TRYS
__RDL:  PUSH    AX      ;SAVE FUNCTION/SECTOR COUNT
        INT     0x13    ;READ/WRITE/VERIFY SECTOR
        POP     AX      ;RESTORE FUNCTION/SECTOR COUNT
        JNC     __X     ;IF NO ERRORS, CONTINUE
        DEC     TRYCNT  ;ELSE, DECREMENT TRY COUNT
        JNZ     __RDL   ;IF NOT LAST TRY, CONTINUE
        MOV     AH,=0   ;ELSE, SET FUNCTION NUMBER=0
        INT     0x13    ;RESET DISK DRIVER
        MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
        RET             ;DONE
__X:    XOR     AL,AL   ;SET RETURN CODE=0
        RET             ;DONE
;
SETUP:  MOV     CH,BYTE PDRTRK[SI]  ;GET TRACK NUMBER
        MOV     CL,BYTE PDRSEC[SI]  ;GET SECTOR NUMBER
        CMP     SECSIZ[SI],=3  ;PHYSICAL SECTOR SIZE=3?
        JZ      __PSS3  ;IF SO, CONTINUE
        MOV     AX,PDRTRK[SI]  ;GET REQUESTED TRACK NUMBER
        MOV     CL,BLKSIZ[SI]  ;GET ALLOCATION BLOCK SIZE
        SHR     AX,CL   ;CALC TRACK NUMBER
        MOV     CH,AL   ;TRACK NUMBER TO CH-REG
        MOV     CL,BLKSIZ[SI]  ;GET ALLOCATION BLOCK SIZE
        NEG     CL      ;NEGATE ALLOCATION BLOCK SIZE
        ADD     CL,=8   ;CALC SHIFT COUNT
        MOV     AL,=0xFF  ;SET AL=0FFH
        SHR     AL,CL   ;CALC ALLOCATION BLOCK MASK
        MOV     CL,BYTE PDRTRK[SI]  ;GET REQUESTED TRACK
        AND     CL,AL   ;CALC SECTOR NUMBER
__PSS3: MOV     BX,PDRDST[SI]  ;GET PDR DST ADDRESS
        CALL    SETNFO  ;SET DISK DRIVER INFO
        MOV     DH,=0   ;PRESET HEAD NUMBER=0
        MOV     AL,VARSPT  ;GET VARIABLE INFO SECTORS/TRACK
        CMP     CL,AL   ;SECTOR ON SIDE ONE?
        JC      __SOS0  ;IF NOT, CONTINUE
        SUB     CL,AL   ;ELSE, CALC SECTOR NUMBER
        INC     DH      ;SET HEAD NUMBER=1
__SOS0: INC     CL      ;INCREMENT SECTOR NUMBER
```

```
        MOV     BX,PDRDMA[SI]  ;GET REQUESTED DMA OFFSET
        MOV     ES,PDRBAS[SI]  ;GET REQUESTED DMA BASE
        MOV     AL,BYTE PDRSC[SI]  ;GET PD REQ SECTOR COUNT
        RET             ;DONE
;
SETNFO: PUSH    BX      ;SAVE CX-REG
        PUSH    CX      ;SAVE CX-REG
        PUSH    SI      ;SAVE SI-REG
        PUSH    DI      ;SAVE DI-REG
        ADD     BX,=VAROFF  ;ADD VARIABLE DRIVER INFO OFFSET
        MOV     SI,BX   ;VARIABLE DISK DRIVER INFO TO SI-REG
        MOV     DI,&VARNFO  ;GET VARIABLE INFO ADDRESS
        MOV     CX,DS   ;GET DS-REG
        MOV     ES,CX   ;SET ES-REG
        MOV     CX,=VARLEN  ;GET VARIABLE INFO LENGTH
        CLD             ;CLEAR DIRECTION FLAG
        REP MOVS BYTE   ;SET VARIABLE DISK DRIVER INFO
        POP     DI      ;RESTORE DI-REG
        POP     SI      ;RESTORE SI-REG
        POP     CX      ;RESTORE CX-REG
        POP     BX      ;RESTORE CX-REG
        RET             ;DONE
;
END
£TITLE  "SAMPLE MEMORY SPECIFICATION TABLE MODULE (256K)"
£SUBTTL "FOR TURBODOS ON IBM PERSONAL COMPUTER"
;
; VERSION: 01/03/84
;
        MODULE  "MSTIPC"  ;MODULE NAME
;
        LOC     Data£   ;LOCATE IN DATA SEGMENT
;
MEMTBL::BYTE    1       ;NUMBER OF MEMORY SEGMENTS
        WORD    0x0050  ;MEMORY SEGMENT BASE
        WORD    0x4000-0x0050  ;MEMORY SEGMENT LENGTH
;
END
```

```
                        ;
                        ; VERSION: 01/05/84
                        ;
0000                            MODULE  "DREQUATE"  ;MODULE NAME
                        ;
                        ; ASCII EQUIVALENCES
                        ;
0000                    ANUL    == BYTE 0x00    ;NULL
0001                    ASOH    == BYTE 0x01    ;SOH
0002                    ASTX    == BYTE 0x02    ;STX
0003                    AETX    == BYTE 0x03    ;ETX
0004                    AEOT    == BYTE 0x04    ;EOT
0005                    AENQ    == BYTE 0x05    ;ENQ
0006                    AACK    == BYTE 0x06    ;ACK
0007                    ABEL    == BYTE 0x07    ;BELL
0008                    ABS     == BYTE 0x08    ;BS
0009                    AHT     == BYTE 0x09    ;HT
000A                    ALF     == BYTE 0x0A    ;LF
000B                    AVT     == BYTE 0x0B    ;VT
000C                    AFF     == BYTE 0x0C    ;FF
000D                    ACR     == BYTE 0x0D    ;CR
000E                    ASO     == BYTE 0x0E    ;SO
000F                    ASI     == BYTE 0x0F    ;SI
0010                    ADLE    == BYTE 0x10    ;DLE
0011                    ADC1    == BYTE 0x11    ;DC1
0012                    ADC2    == BYTE 0x12    ;DC2
0013                    ADC3    == BYTE 0x13    ;DC3
0014                    ADC4    == BYTE 0x14    ;DC4
0015                    ANAK    == BYTE 0x15    ;NAK
0016                    ASYN    == BYTE 0x16    ;SYN
0017                    AETB    == BYTE 0x17    ;ETB
0018                    ACAN    == BYTE 0x18    ;CAN
0019                    AEM     == BYTE 0x19    ;EM
001A                    ASUB    == BYTE 0x1A    ;SUB
001B                    AESC    == BYTE 0x1B    ;ESC
001C                    AFS     == BYTE 0x1C    ;FS
001D                    AGS     == BYTE 0x1D    ;GS
001E                    ARS     == BYTE 0x1E    ;RS
001F                    AUS     == BYTE 0x1F    ;US
0020                    ASP     == BYTE 0x20    ;SPACE
007F                    ARUB    == BYTE 0x7F    ;RUBOUT (DEL)
                        ;
0000                            LOC     0       ;SYMBOLIC DEFINITIONS RELATIVE
                        ;
0000                    PDRDP:                  ;PD REQUEST DESCRIPTOR PACKET
0000                    PDRFCN: RES     BYTE 1  ;PD REQUEST FUNCTION NUMBER
0001                    PDRDRV: RES     BYTE 1  ;PD REQUEST DRIVE NUMBER
0002                    PDRTRK: RES     WORD 1  ;PD REQUEST TRACK NUMBER
0004                    PDRSEC: RES     WORD 1  ;PD REQUEST SECTOR NUMBER
0006                    PDRSC:  RES     WORD 1  ;PD REQUEST SECTOR COUNT
0008                    PDRTC:  RES     WORD 1  ;PD REQUEST TRANSFER COUNT
000A                    PDRDMA: RES     WORD 1  ;PD REQUEST DMA OFFSET
000C                    PDRBAS: RES     WORD 1  ;PD REQUEST DMA BASE
000E                    PDRDST: RES     WORD 1  ;PD REQUEST DRIVE SPEC TABLE A
0010                    PDRLEN  == BYTE .-PDRDP ;PD REQUEST DESCRIPTOR PACKET
0010                    DSKNFO:                 ;DISK TYPE INFORMATION
```

```
0010                    BLKSIZ: RES     BYTE 1  ;BLOCK SIZE
0011                    NMBLKS: RES     WORD 1  ;NUMBER OF BLOCKS
0013                    NMBDIR: RES     BYTE 1  ;NUMBER OF DIRECTORY BLOCKS
0014                    SECSIZ: RES     BYTE 1  ;PHYSICAL SECTOR SIZE (2^N*12(
0015                    SECTRK: RES     WORD 1  ;PHYSICAL SECTORS PER TRACK
0017                    TRKDSK: RES     WORD 1  ;PHYSICAL TRACKS PER DISK
0019                    RESTRK: RES     WORD 1  ;NUMBER OF RESERVED TRACKS
000B                    DNFOL   == BYTE .-DSKNFO  ;DISK INFO LENGTH
                        ;
0000                            RELOC           ;RESTORE PREVIOUS LOCATION
                        ;
0000                            END             ;END OF MODULE
```

```
                        ;
                        ; VERSION: 01/05/84
                        ;
   0000                         MODULE  "MPBIPC"  ;MODULE NAME
                        ;
   0A00                 LODSEG  == 0x0A00        ;LOAD SEGMENT ADDRESS
                        ;
   07C0                 CODE    ==: 0x07C0       ;CODE EXECUTION SEGMENT
                        ;
   0080                 DATALEN == 128           ;DATA WORKING STORAGE LENGTH
                        ;
   0004                 MAXTRY  == 4             ;MAX TRY COUNT
                        ;
   0000                         LOC     0        ;DEFINITIONS RELATIVE TO 0
                        ;
   0000                 DSKNFO:                  ;DISK TYPE INFORMATION
   0000                 BLKSIZ: RES     BYTE 1   ;ALLOCATION BLOCK SIZE
   0001                 NMBLKS: RES     WORD 1   ;NUMBER OF ALLOCATION BLOCKS
   0003                 NMBDIR: RES     BYTE 1   ;NUMBER OF DIRECTORY BLOCKS
   0004                 SECSIZ: RES     BYTE 1   ;PHYSICAL SECTOR SIZE (2^N*128)
   0005                 SECTRK: RES     WORD 1   ;PHYSICAL SECTORS PER TRACK
   0007                 TRKDSK: RES     WORD 1   ;PHYSICAL TRACKS PER DISK
   0009                 RESTRK: RES     WORD 1   ;NUMBER OF RESERVED TRACKS
   000B                 DNFOL   == .-DSKNFO      ;DISK INFO LENGTH
                        ;
   0080                         LOC     DATALEN ;LOCATE AFTER DATA SEGMENT
                        ;
   0080                 WSBASE  == .             ;WORKING STORAGE BASE
                        ;
   0080                 DRIVE:  RES     BYTE 1   ;DRIVE NUMBER
   0081                 TRYCNT: RES     BYTE 1   ;TRY COUNT
                        ;
   0002                 WSLEN   == .-WSBASE      ;WORKING STORAGE LENGTH
                        ;
   07B7                 DATA    ==: CODE-((DATALEN+WSLEN+15)/16)  ;DATA SEGMENT
                        ;
01:0000                         LOC     Code#    ;LOCATE IN CODE SEGMENT
                        ;
01:0000 FB           INIT::  STI              ;ENABLE INTERRUPTS
01:0001 B4 00                MOV     AH,=0    ;SET FUNCTION NUMBER=0
01:0003 CD 13                INT     0x13     ;RESET DISK I/O DRIVER
01:0005 B8 0A00              MOV     AX,&LODSEG  ;GET LOAD SEGMENT ADDRESS
01:0008 C3                   RET              ;DONE
                        ;
01:0009 A2 0080      SELECT::MOV     DRIVE,AL  ;SET DRIVE NUMBER
01:000C 3C 04                CMP     AL,=4    ;VALID DRIVE NUMBER?
01:000E B0 00                MOV     AL,=0    ;PRESET RETURN CODE=0
01:0010 73 14                JNC     __X      ;IF INVALID DRIVE, CONTINUE
01:0012 B9 0008              MOV     CX,=8    ;ELSE, GET SECTOR NUMBER 8
01:0015 B4 04                MOV     AH,=4    ;SET FUNCTION NUMBER=4
01:0017 E8 000F              CALL    RVCOM    ;ATTEMPT TO READ SECTOR 8
01:001A BE 01:008B           MOV     SI,&OSDST  ;PRESET DST ADDRESS=ONE-SIDE
01:001D 84C0                 TEST    AL,AL    ;SECTOR READ SUCCESSFULLY?
01:001F 75 05                JNZ     __X      ;IF NOT, CONTINUE
01:0021 BE 01:0080           MOV     SI,&TSDST  ;ELSE, SET DST ADDRESS=TWO-S
01:0024 F6D0                 NOT     AL       ;SET RETURN CODE=0FFH
```

```
01:0026 C3                  __X:    RET             ;DONE
                            ;
01:0027 B4 02               READ::  MOV     AH,=2   ;SET FUNCTION NUMBER=2
                            ;
01:0029 51                  RVCOM:  PUSH    CX      ;SAVE TRACK NUMBER
01:002A 89CA                        MOV     DX,CX   ;TRACK NUMBER TO DX-REG
01:002C A0 0000                     MOV     AL,BLKSIZ  ;GET ALLOCATION BLOCK SIZE
01:002F 88C1                        MOV     CL,AL   ;SET SHIFT COUNT
01:0031 D3EA                        SHR     DX,CL   ;CALC TRACK NUMBER
01:0033 F6D8                        NEG     AL      ;NEGATE ALLOCATION BLOCK SIZE
01:0035 04 08                       ADD     AL,=8   ;CALC SHIFT COUNT
01:0037 88C1                        MOV     CL,AL   ;SHIFT COUNT TO CL-REG
01:0039 B0 FF                       MOV     AL,=0xFF  ;SET AL=0FFH
01:003B D2E8                        SHR     AL,CL   ;CALC ALLOCATION BLOCK MASK
01:003D 59                          POP     CX      ;RESTORE TRACK NUMBER
01:003E 88D5                        MOV     CH,DL   ;TRACK NUMBER TO CH-REG
01:0040 20C1                        AND     CL,AL   ;CALC SECTOR NUMBER
01:0042 B6 00                       MOV     DH,=0   ;PRESET HEAD NUMBER=0
01:0044 80F9 08                     CMP     CL,=8   ;SECTOR ON SIDE ONE?
01:0047 72 05                       JC      __SOS0  ;IF NOT, CONTINUE
01:0049 80E9 08                     SUB     CL,=8   ;ELSE, CALC SECTOR NUMBER
01:004C FEC6                        INC     DH      ;SET HEAD NUMBER=1
01:004E FEC1                __SOS0: INC     CL      ;INCREMENT SECTOR NUMBER
01:0050 8A16 0080                   MOV     DL,DRIVE  ;GET DRIVE NUMBER
01:0054 B0 01                       MOV     AL,=1   ;SET SECTOR COUNT=1
01:0056 C606 0081 04                MOV     TRYCNT,=MAXTRY  ;SET TRY COUNT=MAX TRY
01:005B 50                  __RDL:  PUSH    AX      ;SAVE FUNCTION/SECTOR COUNT
01:005C CD 13                       INT     0x13    ;READ SECTOR
01:005E 58                          POP     AX      ;RESTORE FUNCTION/SECTOR COUNT
01:005F 73 0D                       JNC     __X     ;IF NO ERRORS, DONE
01:0061 FE0E 0081                   DEC     TRYCNT  ;ELSE, DECREMENT TRY COUNT
01:0065 75 F4                       JNZ     __RDL   ;IF NOT LAST TRY, CONTINUE
01:0067 B4 00                       MOV     AH,=0   ;ELSE, SET FUNCTION NUMBER=0
01:0069 CD 13                       INT     0x13    ;RESET DISK I/O DRIVER
01:006B B0 FF                       MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
01:006D C3                          RET             ;DONE
01:006E 30C0                __X:    XOR     AL,AL   ;SET RETURN CODE=0
01:0070 C3                          RET             ;DONE
                            ;
   0080                     TBUF    == 0x0080       ;DEFAULT BASE PAGE BUFFER
   0100                     TPA     == 0x0100       ;PROGRAM START ADDRESS
   0008                     CMDHDR  == 128/16       ;.CMD HEADER LENGTH (PARAGRAPH
                            ;
01:0071 B8 0A08             XFER::  MOV     AX,&LODSEG+CMDHDR  ;GET LOAD SEGMENT
01:0074 8ED8                        MOV     DS,AX   ;SET DS-REG
01:0076 C606 0080 00                MOV     TBUF,=BYTE 0  ;MAKE DEFAULT BUFFER EMP
01:007B EA 0100 0A08                JMPF    TPA,LODSEG+CMDHDR  ;TRANSFER TO LOADER
                            ;
                            ;       512 BYTE SECTOR, DOUBLE-DENSITY, TWO-SIDED (MI
                            ;
01:0080 04                  TSDST:  BYTE    4       ;BLOCK SIZE
01:0081 00A0                        WORD    (40*8*2*(1<<2))/(1<<4)  ;NUMBER OF BLO
01:0083 02                          BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
01:0084 02                          BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128
01:0085 0001                        WORD    1       ;PHYSICAL SECTORS PER TRACK
01:0087 0280                        WORD    40*8*2  ;PHYSICAL TRACKS PER DISK
```

```
01:0089 0001                    WORD    1       ;RESERVED TRACKS
                        ;
                        ;       512 BYTE SECTOR, DOUBLE-DENSITY, ONE-SIDED (MIN
                        ;
01:008B 03              OSDST:  BYTE    3       ;BLOCK SIZE
01:008C 00A0                    WORD    (40*8*(1<<2))/(1<<3)  ;NUMBER OF BLOCKS
01:008E 02                      BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
01:008F 02                      BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128)
01:0090 0001                    WORD    1       ;PHYSICAL SECTORS PER TRACK
01:0092 0140                    WORD    40*8    ;PHYSICAL TRACKS PER DISK
01:0094 0001                    WORD    1       ;RESERVED TRACKS
                        ;
   0000                 END
```

```
                        ;
                        ; VERSION: 01/03/84
                        ;
   0000                         MODULE  "NITIPC"   ;MODULE NAME
                        ;
                        ; #INCLUDE      "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALEN(
                        ;
01:0000                         LOC     Code#    ;LOCATE IN CODE SEGMENT
                        ;
01:0000 E8 02:0000*     HDWNIT::CALL    RTCNIT# ;INITIALIZE REAL TIME CLOCK DR
01:0003 E8 03:0000*             CALL    CONNIT# ;INITIALIZE CONSOLE DRIVER
01:0006 E8 04:0000*             CALL    DSKINA# ;INITIALIZE DISK DRIVER
01:0009 C3                      RET             ;DONE
                        ;
   0000                 END
```

```
                        ;
                        ; VERSION: 01/03/84
                        ;
   0000                         MODULE  "CONIPC"  ;MODULE NAME
                        ;
                        ; #INCLUDE      "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALEN
                        ;
01:0000                         LOC     Data#   ;LOCATE IN DATA SEGMENT
                        ;
01:0000 0040            CIBSIZ::WORD    64      ;CONSOLE INPUT BUFFER SIZE
01:0002 0000            CIBADR: WORD    0       ;CONSOLE INPUT BUFFER ADDRESS
01:0004 0000            CIIPTR: WORD    0       ;CONSOLE INPUT BUFFER INPUT PC
01:0006 0000            CIOPTR: WORD    0       ;CONSOLE INPUT BUFFER OUTPUT F
01:0008 0000            CIBCNT: WORD    0       ;CONSOLE INPUT BUFFER COUNT
                        ;
01:000A                 CISPH:                  ;CONSOLE INPUT SEMAPHORE
01:000A 0000                    WORD    0       ;SEMAPHORE COUNT
01:000C 01:000C         __CIHD: WORD    __CIHD  ;SEMAPHORE LIST HEAD
01:000E 01:000C                 WORD    __CIHD
                        ;
01:0010 00              CIWCT:  BYTE    0       ;CONSOLE INPUT WAIT COUNT
                        ;
02:0000                         LOC     Code#   ;LOCATE IN CODE SEGMENT
                        ;
02:0000 1E              CONNIT::PUSH    DS      ;SAVE DS-REG
02:0001 31C0                    XOR     AX,AX   ;GET INTERRUPT POINTER PARAGR
02:0003 8ED8                    MOV     DS,AX   ;SET DS-REG
02:0005 C706 0014               MOV     (0x05*4)+0,WORD &PSISR  ;SET VECTOR O
        02:0030
02:000B 8C0E 0016               MOV     (0x05*4)+2,CS  ;SET VECTOR CS BASE AD
02:000F 1F                      POP     DS      ;RESTORE DS-REG
02:0010 B4 00                   MOV     AH,=0   ;SET FUNCTION NUMBER=0
02:0012 B0 07                   MOV     AL,=7   ;SET MODE VALUE=7
02:0014 CD 10                   INT     0x10    ;SET VIDEO MODE=7
02:0016 8B1E 01:0000            MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER
02:001A E8 03:0000*             CALL    ALLOC#  ;ALLOCATE CONSOLE INPUT BUFFE
02:001D 891E 01:0002            MOV     CIBADR,BX  ;SAVE CONSOLE INPUT BUFFER
02:0021 891E 01:0004            MOV     CIIPTR,BX  ;SET CONSOLE INPUT POINTER
02:0025 891E 01:0006            MOV     CIOPTR,BX  ;SET CONSOLE OUTPUT POINTE
02:0029 BA 02:00A5              MOV     DX,&CIPOLL  ;GET CONSOLE INPUT POLL F
02:002C E8 04:0000*             CALL    LNKPOL# ;LINK POLL ROUTINE ON POLL LI
02:002F C3                      RET             ;DONE
                        ;
02:0030 CF              PSISR:  IRET            ;DONE
                        ;
02:0031 80FA 0A         CONDR_::CMP     DL,=10  ;FUNCTION NUMBER=10?
02:0034 74 1D                   JZ      CONOUT  ;IF SO, CONTINUE
02:0036 84D2                    TEST    DL,DL   ;FUNCTION NUMBER=0?
02:0038 74 25                   JZ      CONST   ;IF SO, CONTINUE
02:003A FECA                    DEC     DL      ;FUNCTION NUMBER=1?
02:003C 74 31                   JZ      CONIN   ;IF SO, CONTINUE
02:003E FECA                    DEC     DL      ;FUNCTION NUMBER=2?
02:0040 74 11                   JZ      CONOUT  ;IF SO, CONTINUE
02:0042 80EA 06                 SUB     DL,=8-2 ;FUNCTION NUMBER=8?
02:0045 74 05                   JZ      CONSO   ;IF SO, CONTINUE
02:0047 FECA                    DEC     DL      ;FUNCTION NUMBER=9?
```

```
02:0049 74 01                   JZ      CONSI   ;IF SO, CONTINUE
02:004B C3                      RET             ;ELSE, DONE
                        ;
02:004C                 CONSO:
02:004C E8 05:0000*     CONSI:  CALL    DMS#    ;POSITION TO NEXT LINE
02:004F 0D 0A 00                BYTE    ACR,ALF,0
02:0052 C3                      RET             ;DONE
                        ;
02:0053 BB 0000         CONOUT: MOV     BX,=0   ;SET FOREGROUND COLOR/DISPLAY F
02:0056 88C8                    MOV     AL,CL   ;OUTPUT CHARACTER TO AL-REG
02:0058 B4 0E                   MOV     AH,=14  ;SET FUNCTION NUMBER=14
02:005A CD 10                   INT     0x10    ;OUTPUT CHARACTER TO VIDEO DISF
02:005C B0 FF                   MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
02:005E C3                      RET             ;DONE
                        ;
02:005F A1 01:0008      CONST:  MOV     AX,CIBCNT  ;GET CONSOLE INPUT BUFFER C(
02:0062 85C0                    TEST    AX,AX   ;CONSOLE INPUT BUFFER COUNT=0?
02:0064 74 08                   JZ      __X     ;IF SO, DONE
02:0066 8B1E 01:0006            MOV     BX,CIOPTR  ;ELSE, GET BUFFER OUTPUT PO
02:006A 8A0F                    MOV     CL,[BX] ;GET CONSOLE INPUT CHARACTER
02:006C B0 FF                   MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
02:006E C3              __X:    RET             ;DONE
                        ;
02:006F A1 01:0008      CONIN:  MOV     AX,CIBCNT  ;GET CONSOLE INPUT BUFFER C(
02:0072 85C0                    TEST    AX,AX   ;CONSOLE INPUT BUFFER COUNT=0?
02:0074 74 23                   JZ      __WT    ;IF SO, CONTINUE
02:0076 FF0E 01:0008            DEC     CIBCNT  ;ELSE, DECREMENT INPUT BUFFER
02:007A 8B1E 01:0006            MOV     BX,CIOPTR  ;GET BUFFER OUTPUT POINTER
02:007E 8A07                    MOV     AL,[BX] ;GET CHARACTER FROM INPUT BUFF
02:0080 43                      INC     BX      ;INCREMENT BUFFER OUTPUT POINT
02:0081 89DA                    MOV     DX,BX   ;BUFFER OUTPUT POINTER TO DX-R
02:0083 8B1E 01:0000            MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER S
02:0087 4B                      DEC     BX      ;DECREMENT CONSOLE INPUT BUFFE
02:0088 8B0E 01:0002            MOV     CX,CIBADR  ;GET CONSOLE INPUT BUFFER A
02:008C 01CB                    ADD     BX,CX   ;CALC LAST BUFFER ADDRESS
02:008E 29D3                    SUB     BX,DX   ;BUFFER WRAP-AROUND?
02:0090 73 02                   JNC     __NWA   ;IF NOT, CONTINUE
02:0092 89CA                    MOV     DX,CX   ;GET CONSOLE INPUT BUFFER ADDR
02:0094 8916 01:0006    __NWA:  MOV     CIOPTR,DX  ;UPDATE BUFFER OUTPUT POINT
02:0098 C3                      RET             ;DONE
02:0099 FE06 01:0010    __WT:   INC     CIWCT   ;INCREMENT CONSOLE INPUT WAIT
02:009D BB 01:000A              MOV     BX,&CISPH  ;GET CONSOLE INPUT SEMAPHOR
02:00A0 E8 06:0000*             CALL    WAIT#   ;WAIT FOR CONSOLE INPUT
02:00A3 EB CA                   JMPS    CONIN   ;CONTINUE
                        ;
02:00A5 0000            CIPOLL: WORD    0       ;CONSOLE INPUT POLL ROUTINE
02:00A7 0000                    WORD    0
                        ;
02:00A9 B4 01                   MOV     AH,=1   ;SET KEYBOARD FUNCTION=1
02:00AB CD 16                   INT     0x16    ;GET KEYBOARD STATUS
02:00AD 74 63                   JZ      __X     ;IF NO CHARACTER AVAILABLE, CC
02:00AF B4 00                   MOV     AH,=0   ;ELSE, SET KEYBOARD FUNCTION=0
02:00B1 CD 16                   INT     0x16    ;GET KEYBOARD INPUT
02:00B3 3D 5300                 CMP     AX,=83<<8  ;DELETE KEY PRESSED?
02:00B6 75 02                   JNZ     __NDEL  ;IF NOT, CONTINUE
02:00B8 B0 7F                   MOV     AL,=ARUB  ;SUBSTITUTE ASCII RUBOUT COI
```

```
02:00BA 88C1              __NDEL: MOV     CL,AL    ;CONSOLE INPUT CHARACTER TO CL-
02:00BC 80E1 7F                   AND     CL,=0x7F  ;STRIP SIGN BIT ON CHARACTER
02:00BF A0 07:0000                MOV     AL,ATNCHR#  ;GET ATTENTION CHARACTER
02:00C2 38C8                      CMP     AL,CL    ;CHARACTER=ATTENTION CHARACTER?
02:00C4 75 0C                     JNZ     __NACH   ;IF NOT, CONTINUE
02:00C6 A1 01:0004                MOV     AX,CIIPTR  ;ELSE, GET BUFFER INPUT POI
02:00C9 A3 01:0006                MOV     CIOPTR,AX  ;SET BUFFER OUTPUT POINTER
02:00CC C706 01:0008              MOV     CIBCNT,=0  ;SET CONSOLE INPUT BUFFER CC
       0000
02:00D2 8B1E 01:0000      __NACH: MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER SI
02:00D6 8B16 01:0008              MOV     DX,CIBCNT  ;GET CONSOLE INPUT BUFFER CC
02:00DA 42                        INC     DX       ;INCREMENT CONSOLE INPUT COUNT
02:00DB 29D3                      SUB     BX,DX    ;CONSOLE INPUT BUFFER FULL?
02:00DD 72 33                     JC      __X      ;IF SO, DONE
02:00DF 8916 01:0008              MOV     CIBCNT,DX  ;ELSE, UPDATE CONSOLE INPUT
02:00E3 8B1E 01:0004              MOV     BX,CIIPTR  ;GET BUFFER INPUT POINTER
02:00E7 880F                      MOV     [BX],CL ;STORE INPUT CHARACTER IN BUFFE
02:00E9 43                        INC     BX       ;INCREMENT BUFFER INPUT POINTE
02:00EA 89DA                      MOV     DX,BX    ;BUFFER INPUT POINTER TO DX-REC
02:00EC 8B1E 01:0000              MOV     BX,CIBSIZ  ;GET CONSOLE INPUT BUFFER SI
02:00F0 4B                        DEC     BX       ;DECREMENT CONSOLE INPUT BUFFE
02:00F1 8B0E 01:0002              MOV     CX,CIBADR  ;GET CONSOLE INPUT BUFFER AI
02:00F5 01CB                      ADD     BX,CX    ;CALC LAST BUFFER ADDRESS
02:00F7 29D3                      SUB     BX,DX    ;BUFFER WRAP-AROUND?
02:00F9 73 02                     JNC     __NWA    ;IF NOT, CONTINUE
02:00FB 89CA                      MOV     DX,CX    ;GET CONSOLE INPUT BUFFER ADDR
02:00FD 8916 01:0004      __NWA:  MOV     CIIPTR,DX  ;UPDATE BUFFER INPUT POINTE
02:0101 BB 01:0010                MOV     BX,&CIWCT  ;GET CONSOLE INPUT WAIT COU
02:0104 8A07                      MOV     AL,[BX] ;GET SERIAL INPUT WAIT COUNT
02:0106 84C0                      TEST    AL,AL    ;SERIAL INPUT WAIT COUNT=0?
02:0108 74 08                     JZ      __X      ;IF SO, DONE
02:010A FE0F                      DEC     BYTE [BX]  ;ELSE, DECREMENT INPUT WAIT
02:010C BB 01:000A                MOV     BX,&CISPH  ;GET CONSOLE INPUT SEMAPHOR
02:010F E8 08:0000*               CALL    SIGNAL# ;SIGNAL PROCESS AS READY
02:0112 C3                __X:    RET              ;DONE
                          ;
   0000                   END
```

```
                        ;
                        ; VERSION: 01/04/84
                        ;
   0000                         MODULE  "LSTPPA"  ;MODULE NAME
                        ;
                        ; #INCLUDE       "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALEN
                        ;
01:0000                         LOC     Data#   ;LOCATE IN DATA SEGMENT
                        ;
01:0000 00              PPACH:: BYTE    0       ;PARALLEL PRINTER CHANNEL NUM
01:0001 00              INITC:  BYTE    0       ;INITIALIZATION COMPLETE FLAG
01:0002 00              COLCNT: BYTE    0       ;COLUMN COUNTER
01:0003 00              SPCNT:  BYTE    0       ;SPACE COUNTER
01:0004 00              LNCNT:  BYTE    0       ;LINE COUNTER
                        ;
01:0005                 LOSPH:                  ;LIST OUTPUT SEMAPHORE
01:0005 0000                    WORD    0       ;SEMAPHORE COUNT
01:0007 01:0007         __LOHD: WORD    __LOHD  ;SEMAPHORE LIST HEAD
01:0009 01:0007                 WORD    __LOHD
                        ;
02:0000                         LOC     Code#   ;LOCATE IN DATA SEGMENT
                        ;
02:0000 A0 01:0001      LSTDR_::MOV     AL,INITC  ;GET INITIALIZATION COMPLET
02:0003 84C0                    TEST    AL,AL   ;INITIALIZATION COMPLETE FLAG
02:0005 75 11                   JNZ     __LDRV  ;IF SO, CONTINUE
02:0007 C606 01:0001 FF         MOV     INITC,=0xFF  ;ELSE, SET INIT COMPLETE
02:000C 52                      PUSH    DX      ;SAVE FUNCTION NUMBER
02:000D 8A16 01:0000            MOV     DL,PPACH  ;GET PRINTER CHANNEL NUMBER
02:0011 B6 00                   MOV     DH,=0   ;DOUBLE LENGTH
02:0013 B4 01                   MOV     AH,=1   ;SET FUNCTION NUMBER=1
02:0015 CD 17                   INT     0x17    ;INITIALIZE PRINTER PORT
02:0017 5A                      POP     DX      ;RESTORE FUNCTION NUMBER
02:0018 80FA 02         __LDRV: CMP     DL,=2   ;FUNCTION NUMBER=2?
02:001B 74 1E                   JZ      LSTOUT  ;IF SO, CONTINUE
02:001D 80FA 07                 CMP     DL,=7   ;FUNCTION NUMBER=7?
02:0020 74 01                   JZ      LSTWSR  ;IF SO, CONTINUE
02:0022 C3                      RET             ;ELSE, DONE
                        ;
02:0023 A0 01:0002      LSTWSR: MOV     AL,COLCNT  ;GET COLUMN COUNT
02:0026 50                      PUSH    AX      ;SAVE COLUMN COUNT
02:0027 B1 0D                   MOV     CL,=ACR ;GET CARRIAGE RETURN
02:0029 E8 000F                 CALL    LSTOUT  ;OUTPUT CARRIAGE RETURN
02:002C 58                      POP     AX      ;RESTORE COLUMN COUNT
02:002D 84C0                    TEST    AL,AL   ;COLUMN COUNT=0?
02:002F 75 08                   JNZ     __FF    ;IF NOT, CONTINUE
02:0031 A0 01:0004              MOV     AL,LNCNT  ;ELSE, GET LINE COUNTER
02:0034 84C0                    TEST    AL,AL   ;LINE COUNT=0?
02:0036 75 01                   JNZ     __FF    ;IF NOT, CONTINUE
02:0038 C3                      RET             ;ELSE, DONE
02:0039 B1 0C           __FF:   MOV     CL,=AFF ;GET FORM FEED
                        ;
02:003B 88C8            LSTOUT: MOV     AL,CL   ;GET OUTPUT CHARACTER
02:003D 24 7F                   AND     AL,=0x7F  ;STRIP PARITY
02:003F 74 4D                   JZ      __X     ;IF CHARACTER=NULL, DONE
02:0041 88C1                    MOV     CL,AL   ;ELSE, CHARACTER TO C-REG
02:0043 3C 20                   CMP     AL,=ASP ;GRAPHIC CHARACTER?
```

```
02:0045 73 69                    JNC     __GR    ;IF SO, PROCESS
02:0047 3C 0D                    CMP     AL,=ACR ;CARRIAGE RETURN?
02:0049 74 18                    JZ      __CR    ;IF SO, PROCESS
02:004B 3C 0C                    CMP     AL,=AFF ;FORM FEED?
02:004D 74 20                    JZ      __FF    ;IF SO, PROCESS
02:004F 3C 0A                    CMP     AL,=ALF ;LINE FEED?
02:0051 74 2A                    JZ      __LF    ;IF SO, PROCESS
02:0053 3C 09                    CMP     AL,=AHT ;HORIZONTAL TAB?
02:0055 74 38                    JZ      __HT    ;IF SO, PROCESS
02:0057 3C 08                    CMP     AL,=ABS ;BACKSPACE?
02:0059 74 4D                    JZ      __BS    ;IF SO, CONTINUE
02:005B 3C 07                    CMP     AL,=ABEL  ;BELL?
02:005D 75 2F                    JNZ     __X     ;IF NOT, CONTINUE
02:005F E9 0076                  JMP     LSTCOM  ;ELSE, CONTINUE
02:0062 C3                       RET             ;DONE
02:0063 E8 0072          __CR:   CALL    LSTCOM  ;OUTPUT CARRIAGE RETURN
02:0066 30C0                     XOR     AL,AL
02:0068 A2 01:0002               MOV     COLCNT,AL  ;SET COLUMN COUNT=0
02:006B A2 01:0003               MOV     SPCNT,AL  ;SET SPACE COUNT=0
02:006E C3                       RET             ;DONE
02:006F B1 0A            __FF:   MOV     CL,=ALF ;GET LINE FEED CHARACTER
02:0071 E8 0009                  CALL    __LF    ;OUTPUT LINE FEED
02:0074 A0 01:0004               MOV     AL,LNCNT  ;GET LINE COUNTER
02:0077 84C0                     TEST    AL,AL   ;AT TOP OF FORM YET?
02:0079 74 13                    JZ      __X     ;IF SO, DONE
02:007B EB F2                    JMPS    __FF    ;ELSE, CONTINUE
02:007D E8 0058          __LF:   CALL    LSTCOM  ;OUTPUT LINE FEED
02:0080 A0 01:0004               MOV     AL,LNCNT  ;GET LINE COUNTER
02:0083 FEC0                     INC     AL      ;INCREMENT IT
02:0085 3C 42                    CMP     AL,=66  ;AT TOP OF FORM YET?
02:0087 72 02                    JC      __ULC   ;IF NOT, UDATE LINE COUNT
02:0089 30C0                     XOR     AL,AL   ;ELSE ,RESET LINE COUNTER
02:008B A2 01:0004       __ULC:  MOV     LNCNT,AL  ;UPDATE LINE COUNTER
02:008E C3               __X:    RET             ;DONE
02:008F A0 01:0002       __HT:   MOV     AL,COLCNT  ;GET COLUMN COUNTER
02:0092 88C6                     MOV     DH,AL   ;TO DH-REG
02:0094 24 F8                    AND     AL,=~7  ;CALC NEXT TAB STOP
02:0096 04 08                    ADD     AL,=8
02:0098 28F0                     SUB     AL,DH   ;CALC NUMBER OF SPACES REQUIRE
02:009A 88C6                     MOV     DH,AL   ;SPACE COUNT TO DH-REG
02:009C B1 20            __HTL:  MOV     CL,=ASP ;GET ASCII SPACE
02:009E 88C8                     MOV     AL,CL   ;ASCII SPACE TO A-REG
02:00A0 E8 000D                  CALL    __GR    ;OUTPUT SPACES
02:00A3 FECE                     DEC     DH      ;TO NEXT TAB STOP
02:00A5 75 F5                    JNZ     __HTL
02:00A7 C3                       RET             ;DONE
02:00A8 E8 002D          __BS:   CALL    LSTCOM  ;OUTPUT BACKSPACE
02:00AB FE0E 01:0002             DEC     COLCNT  ;DECREMENT COLUMN COUNT
02:00AF C3                       RET             ;DONE
02:00B0 FE06 01:0002     __GR:   INC     COLCNT  ;INCREMENT COLUMN COUNT
02:00B4 3C 20                    CMP     AL,=ASP ;CHARACTER=SPACE?
02:00B6 74 1B                    JZ      __SP    ;IF SO, CONTINUE
02:00B8 A0 01:0003               MOV     AL,SPCNT  ;ELSE, GET SPACE COUNT
02:00BB 84C0                     TEST    AL,AL   ;SPACE COUNT=0?
02:00BD 74 19                    JZ      LSTCOM  ;IF SO, CONTINUE
02:00BF 88C6                     MOV     DH,AL   ;ELSE, SPACE COUNT TO DH-REG
```

```
'PA: SAMPLE PRINTER DRIVER MODULE (PARALLEL PRINTER ADAPTER)
 TURBODOS ON IBM PERSONAL COMPUTER
: 12

)0C1 51                         PUSH    CX      ;SAVE OUTPUT CHARACTER
)0C2 B1 20              __SPL:  MOV     CL,=ASP ;GET ASCII SPACE
)0C4 E8 0011                    CALL    LSTCOM  ;OUTPUT SPACES
)0C7 FECE                       DEC     DH
)0C9 75 F7                      JNZ     __SPL
)0CB 59                         POP     CX      ;RESTORE OUTPUT CHARACTER
)0CC 30C0                       XOR     AL,AL
)0CE A2 01:0003                 MOV     SPCNT,AL  ;SET SPACE COUNT=0
)0D1 EB 05                      JMPS    LSTCOM  ;CONTINUE
)0D3 FE06 01:0003       __SP:   INC     SPCNT   ;INCREMENT SPACE COUNT
)0D7 C3                         RET             ;DONE
                        ;
)0D8 52                 LSTCOM: PUSH    DX      ;SAVE DX-REG
)0D9 51                         PUSH    CX      ;SAVE CX-REG
)0DA BA 02:00F8                 MOV     DX,&__LOPR  ;GET LIST OUTPUT POLL ROUTINE
)0DD E8 03:0000*                CALL    LNKPOL# ;LINK POLL ROUTINE ON POLL LIST
)0E0 E8 0019                    CALL    __LOPC  ;EXECUTE POLL ROUTINE CODE
)0E3 BB 01:0005                 MOV     BX,&LOSPH  ;GET SEMAPHORE ADDRESS
)0E6 E8 04:0000*                CALL    WAIT#   ;DISPATCH, IF NECESSARY
)0E9 59                         POP     CX      ;RESTORE CX-REG
)0EA 8A16 01:0000               MOV     DL,PPACH  ;GET PRINTER CHANNEL NUMBER
)0EE B6 00                      MOV     DH,=0   ;DOUBLE LENGTH
)0F0 88C8                       MOV     AL,CL   ;GET OUTPUT CHARACTER
)0F2 B4 00                      MOV     AH,=0   ;SET FUNCTION NUMBER=0
)0F4 CD 17                      INT     0x17    ;OUTPUT CHARACTER
)0F6 5A                         POP     DX      ;RESTORE DX-REG
)0F7 C3                         RET             ;DONE
)0F8 0000               __LOPR: WORD    0       ;SUCCESSOR LINK POINTER
)0FA 0000                       WORD    0       ;PREDECESSOR LINK POINTER
)0FC 8A16 01:0000       __LOPC: MOV     DL,PPACH  ;GET PRINTER CHANNEL NUMBER
0100 B6 00                      MOV     DH,=0   ;DOUBLE LENGTH
0102 B4 02                      MOV     AH,=2   ;SET FUNCTION NUMBER=2
0104 CD 17                      INT     0x17    ;GET PRINTER PORT STATUS
0106 F6C4 80                    TEST    AH,=1<<7  ;PRINTER NOT BUSY BIT SET?
0109 74 0C                      JZ      __LOPX  ;IF NOT, CONTINUE
010B BB 02:00F8                 MOV     BX,&__LOPR  ;GET LIST OUTPUT POLL ROUTINE
010E E8 05:0000*                CALL    UNLINK# ;UNLINK POLL ROUTINE FROM POLL LIS
0111 BB 01:0005                 MOV     BX,&LOSPH  ;GET SEMAPHORE ADDRESS
0114 E8 06:0000*                CALL    SIGNAL# ;SIGNAL PROCESS AS READY
0117 C3                 __LOPX: RET             ;DONE
                        ;
0000                    END
```

```
                        ;
                        ; VERSION: 01/04/84
                        ;
   0000                         MODULE  "LSTACA"  ;MODULE NAME
                        ;
                        ; #INCLUDE      "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALENC
                        ;
01:0000                         LOC     Data#   ;LOCATE IN DATA SEGMENT
                        ;
01:0000 67              ACAIP:: BYTE    0x67    ;ASYNC COMM ADAPTER INIT PARAME
01:0001 00              ACACH:: BYTE    0       ;ASYNC COMM ADAPTER CHANNEL NUM
01:0002 00              INITC:  BYTE    0       ;INITIALIZATION COMPLETE FLAG
01:0003 00              COLCNT: BYTE    0       ;COLUMN COUNTER
01:0004 00              SPCNT:  BYTE    0       ;SPACE COUNTER
01:0005 00              LNCNT:  BYTE    0       ;LINE COUNTER
                        ;
01:0006                 COSPH:                  ;COMM OUTPUT SEMAPHORE
01:0006 0000                    WORD    0       ;SEMAPHORE COUNT
01:0008 01:0008         __COHD: WORD    __COHD  ;SEMAPHORE LIST HEAD
01:000A 01:0008                 WORD    __COHD
                        ;
02:0000                         LOC     Code#   ;LOCATE IN DATA SEGMENT
                        ;
02:0000 A0 01:0002      LSTDR_::MOV     AL,INITC  ;GET INITIALIZATION COMPLETE
02:0003 84C0                    TEST    AL,AL   ;INITIALIZATION COMPLETE FLAG S
02:0005 75 14                   JNZ     __LDRV  ;IF SO, CONTINUE
02:0007 C606 01:0002 FF         MOV     INITC,=0xFF  ;ELSE, SET INIT COMPLETE F
02:000C 52                      PUSH    DX      ;SAVE FUNCTION NUMBER
02:000D 8A16 01:0001            MOV     DL,ACACH  ;GET COMM ADAPTER CHANNEL NUM
02:0011 B6 00                   MOV     DH,=0   ;DOUBLE LENGTH
02:0013 A0 01:0000              MOV     AL,ACAIP  ;GET INIT PARAMETERS
02:0016 B4 00                   MOV     AH,=0   ;SET FUNCTION NUMBER=0
02:0018 CD 14                   INT     0x14    ;SET CHANNEL BUAD RATE
02:001A 5A                      POP     DX      ;RESTORE FUNCTION NUMBER
02:001B 80FA 02         __LDRV: CMP     DL,=2   ;FUNCTION NUMBER=2?
02:001E 74 1E                   JZ      LSTOUT  ;IF SO, CONTINUE
02:0020 80FA 07                 CMP     DL,=7   ;FUNCTION NUMBER=7?
02:0023 74 01                   JZ      LSTWSR  ;IF SO, CONTINUE
02:0025 C3                      RET             ;ELSE, DONE
                        ;
02:0026 A0 01:0003      LSTWSR: MOV     AL,COLCNT  ;GET COLUMN COUNT
02:0029 50                      PUSH    AX      ;SAVE COLUMN COUNT
02:002A B1 0D                   MOV     CL,=ACR ;GET CARRIAGE RETURN
02:002C E8 000F                 CALL    LSTOUT  ;OUTPUT CARRIAGE RETURN
02:002F 58                      POP     AX      ;RESTORE COLUMN COUNT
02:0030 84C0                    TEST    AL,AL   ;COLUMN COUNT=0?
02:0032 75 08                   JNZ     __FF    ;IF NOT, CONTINUE
02:0034 A0 01:0005              MOV     AL,LNCNT  ;ELSE, GET LINE COUNTER
02:0037 84C0                    TEST    AL,AL   ;LINE COUNT=0?
02:0039 75 01                   JNZ     __FF    ;IF NOT, CONTINUE
02:003B C3                      RET             ;ELSE, DONE
02:003C B1 0C           __FF:   MOV     CL,=AFF ;GET FORM FEED
                        ;
02:003E 88C8            LSTOUT: MOV     AL,CL   ;GET OUTPUT CHARACTER
02:0040 24 7F                   AND     AL,=0x7F  ;STRIP PARITY
02:0042 74 60                   JZ      __X     ;IF CHARACTER=NULL, DONE
```

```
02:0044 88C1                    MOV     CL,AL   ;ELSE, CHARACTER TO C-REG
02:0046 3C 20                   CMP     AL,=ASP ;GRAPHIC CHARACTER?
02:0048 73 7C                   JNC     __GR    ;IF SO, PROCESS
02:004A 3C 0D                   CMP     AL,=ACR ;CARRIAGE RETURN?
02:004C 74 18                   JZ      __CR    ;IF SO, PROCESS
02:004E 3C 0C                   CMP     AL,=AFF ;FORM FEED?
02:0050 74 2E                   JZ      __FF    ;IF SO, PROCESS
02:0052 3C 0A                   CMP     AL,=ALF ;LINE FEED?
02:0054 74 38                   JZ      __LF    ;IF SO, PROCESS
02:0056 3C 09                   CMP     AL,=AHT ;HORIZONTAL TAB?
02:0058 74 4B                   JZ      __HT    ;IF SO, PROCESS
02:005A 3C 08                   CMP     AL,=ABS ;BACKSPACE?
02:005C 74 60                   JZ      __BS    ;IF SO, CONTINUE
02:005E 3C 07                   CMP     AL,=ABEL  ;BELL?
02:0060 75 42                   JNZ     __X     ;IF NOT, CONTINUE
02:0062 E9 0095                 JMP     COMOUT  ;ELSE, CONTINUE
02:0065 C3                      RET             ;DONE
02:0066 E8 0091         __CR:   CALL    COMOUT  ;OUTPUT CARRIAGE RETURN
02:0069 A0 01:0003              MOV     AL,COLCNT  ;GET COLUMN COUNT
02:006C D0E8                    SHR     AL,=1   ;/2
02:006E D0E8                    SHR     AL,=1   ;/2=/4
02:0070 D0E8                    SHR     AL,=1   ;/2=/8
02:0072 04 03                   ADD     AL,=3   ;ADD CONSTANT
02:0074 E8 0077                 CALL    __NUL   ;OUTPUT NULLS
02:0077 30C0                    XOR     AL,AL
02:0079 A2 01:0003              MOV     COLCNT,AL  ;SET COLUMN COUNT=0
02:007C A2 01:0004              MOV     SPCNT,AL  ;SET SPACE COUNT=0
02:007F C3                      RET             ;DONE
02:0080 B1 0A           __FF:   MOV     CL,=ALF ;GET LINE FEED CHARACTER
02:0082 E8 0009                 CALL    __LF    ;OUTPUT LINE FEED
02:0085 A0 01:0005              MOV     AL,LNCNT  ;GET LINE COUNTER
02:0088 84C0                    TEST    AL,AL   ;AT TOP OF FORM YET?
02:008A 74 18                   JZ      __X     ;IF SO, DONE
02:008C EB F2                   JMPS    __FF    ;ELSE, CONTINUE
02:008E E8 0069         __LF:   CALL    COMOUT  ;OUTPUT LINE FEED
02:0091 B0 02                   MOV     AL,=2   ;GET NULL COUNT
02:0093 E8 0058                 CALL    __NUL   ;OUTPUT NULLS
02:0096 A0 01:0005              MOV     AL,LNCNT  ;GET LINE COUNTER
02:0099 FEC0                    INC     AL      ;INCREMENT IT
02:009B 3C 42                   CMP     AL,=66  ;AT TOP OF FORM YET?
02:009D 72 02                   JC      __ULC   ;IF NOT, UDATE LINE COUNT
02:009F 30C0                    XOR     AL,AL   ;ELSE ,RESET LINE COUNTER
02:00A1 A2 01:0005      __ULC:  MOV     LNCNT,AL  ;UPDATE LINE COUNTER
02:00A4 C3              __X:    RET             ;DONE
02:00A5 A0 01:0003      __HT:   MOV     AL,COLCNT  ;GET COLUMN COUNTER
02:00A8 88C6                    MOV     DH,AL   ;TO DH-REG
02:00AA 24 F8                   AND     AL,=~7  ;CALC NEXT TAB STOP
02:00AC 04 08                   ADD     AL,=8
02:00AE 28F0                    SUB     AL,DH   ;CALC NUMBER OF SPACES REQUIR
02:00B0 88C6                    MOV     DH,AL   ;SPACE COUNT TO DH-REG
02:00B2 B1 20           __HTL:  MOV     CL,=ASP ;GET ASCII SPACE
02:00B4 88C8                    MOV     AL,CL   ;ASCII SPACE TO A-REG
02:00B6 E8 000D                 CALL    __GR    ;OUTPUT SPACES
02:00B9 FECE                    DEC     DH      ;TO NEXT TAB STOP
02:00BB 75 F5                   JNZ     __HTL
02:00BD C3                      RET             ;DONE
```

```
02:00BE E8 0039          __BS:   CALL    COMOUT  ;OUTPUT BACKSPACE
02:00C1 FE0E 01:0003             DEC     COLCNT  ;DECREMENT COLUMN COUNT
02:00C5 C3                       RET             ;DONE
02:00C6 FE06 01:0003     __GR:   INC     COLCNT  ;INCREMENT COLUMN COUNT
02:00CA 3C 20                    CMP     AL,=ASP ;CHARACTER=SPACE?
02:00CC 74 1B                    JZ      __SP    ;IF SO, CONTINUE
02:00CE A0 01:0004               MOV     AL,SPCNT  ;ELSE, GET SPACE COUNT
02:00D1 84C0                     TEST    AL,AL   ;SPACE COUNT=0?
02:00D3 74 25                    JZ      COMOUT  ;IF SO, CONTINUE
02:00D5 88C6                     MOV     DH,AL   ;ELSE, SPACE COUNT TO DH-REG
02:00D7 51                       PUSH    CX      ;SAVE OUTPUT CHARACTER
02:00D8 B1 20            __SPL:  MOV     CL,=ASP ;GET ASCII SPACE
02:00DA E8 001D                  CALL    COMOUT  ;OUTPUT SPACES
02:00DD FECE                     DEC     DH
02:00DF 75 F7                    JNZ     __SPL
02:00E1 59                       POP     CX      ;RESTORE OUTPUT CHARACTER
02:00E2 30C0                     XOR     AL,AL
02:00E4 A2 01:0004               MOV     SPCNT,AL  ;SET SPACE COUNT=0
02:00E7 EB 11                    JMPS    COMOUT  ;CONTINUE
02:00E9 FE06 01:0004     __SP:   INC     SPCNT   ;INCREMENT SPACE COUNT
02:00ED C3                       RET             ;DONE
02:00EE 88C6             __NUL:  MOV     DH,AL   ;NULL COUNT TO DH-REG
02:00F0 B1 00                    MOV     CL,=0   ;SET OUTPUT CHARACTER=NULL
02:00F2 E8 0005          __NULL: CALL    COMOUT  ;OUTPUT NULL
02:00F5 FECE                     DEC     DH      ;DECREMENT NULL COUNT
02:00F7 75 F9                    JNZ     __NULL  ;CONTINUE
02:00F9 C3                       RET             ;DONE
                         ;
02:00FA 52               COMOUT: PUSH    DX      ;SAVE DX-REG
02:00FB 51                       PUSH    CX      ;SAVE CX-REG
02:00FC BA 02:011A               MOV     DX,&__COPR  ;GET COMM OUTPUT POLL ROUTI
02:00FF E8 03:0000*              CALL    LNKPOL# ;LINK POLL ROUTINE ON POLL LIST
02:0102 E8 0019                  CALL    __COPC  ;EXECUTE POLL ROUTINE CODE
02:0105 BB 01:0006               MOV     BX,&COSPH  ;GET SEMAPHORE ADDRESS
02:0108 E8 04:0000*              CALL    WAIT#   ;DISPATCH, IF NECESSARY
02:010B 59                       POP     CX      ;RESTORE CX-REG
02:010C 8A16 01:0001             MOV     DL,ACACH  ;GET COMM ADAPTER CHANNEL NUM
02:0110 B6 00                    MOV     DH,=0   ;DOUBLE LENGTH
02:0112 88C8                     MOV     AL,CL   ;GET OUTPUT CHARACTER
02:0114 B4 01                    MOV     AH,=1   ;SET FUNCTION NUMBER=1
02:0116 CD 14                    INT     0x14    ;OUTPUT CHARACTER
02:0118 5A                       POP     DX      ;RESTORE DX-REG
02:0119 C3                       RET             ;DONE
02:011A 0000             __COPR: WORD    0       ;SUCCESSOR LINK POINTER
02:011C 0000                     WORD    0       ;PREDECESSOR LINK POINTER
02:011E 8A16 01:0001     __COPC: MOV     DL,ACACH  ;GET COMM ADAPTER CHANNEL NUM
02:0122 B6 00                    MOV     DH,=0   ;DOUBLE LENGTH
02:0124 B4 03                    MOV     AH,=3   ;SET FUNCTION NUMBER=3
02:0126 CD 14                    INT     0x14    ;GET COMM CHANNEL STATUS
02:0128 F6C4 20                  TEST    AH,=1<<5  ;TRANSMIT HOLDING REGISTER EM
02:012B 74 0C                    JZ      __COPX  ;IF NOT, CONTINUE
02:012D BB 02:011A               MOV     BX,&__COPR  ;GET COMM OUTPUT POLL ROUTI
02:0130 E8 05:0000*              CALL    UNLINK# ;UNLINK POLL ROUTINE FROM POLL
02:0133 BB 01:0006               MOV     BX,&COSPH  ;GET SEMAPHORE ADDRESS
02:0136 E8 06:0000*              CALL    SIGNAL# ;SIGNAL PROCESS AS READY
02:0139 C3               __COPX: RET             ;DONE
```

```
                                ;
0000                            END
```

```
                        ;
                        ; VERSION: 01/03/84
                        ;
   0000                         MODULE  "RTCIPC"  ;MODULE NAME
                        ;
                        ; #INCLUDE      "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALENC
                        ;
01:0000                         LOC     Data#   ;LOCATE IN DATA SEGMENT
                        ;
01:0000 00              TICACC: BYTE    0       ;TICK ACCUMULATOR
01:0001 00              TICCTR: BYTE    0       ;TICK COUNTER
                        ;
02:0000                         LOC     Code#   ;LOCATE IN CODE SEGMENT
                        ;
02:0000 1E              RTCNIT::PUSH    DS      ;SAVE DS-REG
02:0001 31C0                    XOR     AX,AX   ;GET INTERRUPT POINTER PARAGRAI
02:0003 8ED8                    MOV     DS,AX   ;SET DS-REG
02:0005 C706 0070               MOV     (0x1C*4)+0,WORD &RTCISR  ;SET VECTOR OI
        02:0011
02:000B 8C0E 0072               MOV     (0x1C*4)+2,CS  ;SET VECTOR CS BASE ADDI
02:000F 1F                      POP     DS      ;RESTORE DS-REG
02:0010 C3                      RET             ;DONE
                        ;
02:0011 9D              RTCISR: POPF            ;CLEAR STACK (IP-REG)
02:0012 9D                      POPF            ;CLEAR STACK (CS-REG)
02:0013 9D                      POPF            ;CLEAR STACK (PS-REG)
02:0014 53                      PUSH    BX      ;SAVE REGISTERS
02:0015 51                      PUSH    CX
02:0016 E8 03:0000*             CALL    GETSDS# ;GET SYSTEM DATA SEGMENT
02:0019 FE06 01:0000            INC     TICACC  ;INCREMENT TICK ACCUMULATED
02:001D 803E 01:0000 57         CMP     TICACC,=87  ;EXTRA TICK ACCUMULATED?
02:0022 72 09                   JC      __NETA  ;IF NOT, CONTINUE
02:0024 C606 01:0000 00         MOV     TICACC,=0  ;ELSE, RESET TICK ACCUMULATC
02:0029 FE0E 01:0001            DEC     TICCTR  ;DECREMENT TICK COUNTER
02:002D FE06 01:0001    __NETA: INC     TICCTR  ;INCREMENT TICK COUNTER
02:0031 803E 01:0001 12         CMP     TICCTR,=18  ;ONE SECOND ELAPSED?
02:0036 72 08                   JC      __NSEC  ;IF NOT, CONTINUE
02:0038 C606 01:0001 00         MOV     TICCTR,=0  ;ELSE, RESET TICK COUNTER
02:003D E8 04:0000*             CALL    RTCSEC# ;ADVANCE SYSTEM TIME SECONDS
02:0040 E8 05:0000*     __NSEC: CALL    DLYTIC# ;ADVANCE SYSTEM TICK COUNTER
02:0043 B0 20                   MOV     AL,=0x20  ;GET END OF INTERRUPT COMMANI
02:0045 E6 20                   OUT     0x20,AL ;SIGNAL END OF INTERRUPT
02:0047 59                      POP     CX      ;RESTORE REGISTERS
02:0048 5B                      POP     BX
02:0049 5A                      POP     DX
02:004A 58                      POP     AX
02:004B 1F                      POP     DS
02:004C E9 06:0000*             JMP     ISRXIT# ;CONTINUE
                        ;
   0000                 END
```

```
                        ;
                        ; VERSION: 01/03/84
                        ;
   0000                         MODULE  "DSKIPC"  ;MODULE NAME
                        ;
                        ; #INCLUDE      "DREQUATE"  ;DRIVER SYMBOLIC EQUIVALEN
                        ;
   0004                 MAXTRY  == 4            ;MAX TRY COUNT
                        ;
01:0000                         LOC     Data#   ;LOCATE IN DATA SEGMENT
                        ;
01:0000                 FCNTBL:                 ;FUNCTION PROCESSOR ADDRESS TA
01:0000 02:0036                 WORD    RDDSK   ;READ DISK
01:0002 02:003D                 WORD    WRDSK   ;WRITE DISK
01:0004 02:0074                 WORD    RETDST  ;RETURN DST ADDRESS
01:0006 02:00AF                 WORD    RETRDY  ;RETURN READY STATUS
01:0008 02:0044                 WORD    FMTDSK  ;FORMAT DISK
                        ;
   0005                 NMBFCN  == (.-FCNTBL)/2 ;NUMBER OF FUNCTION PROCESSORS
                        ;
01:000A                 DMXSPH:                 ;MUTUAL EXCLUSION SEMAPHORE
01:000A 0001                    WORD    1       ;SEMAPHORE COUNT
01:000C 01:000C         __DMXH: WORD    __DMXH  ;SEMAPHORE P/D HEAD
01:000E 01:000C                 WORD    __DMXH
                        ;
01:0010 00              TRYCNT: BYTE    0       ;TRY COUNT
                        ;
01:0011                 DRVNFO:                 ;DISK DRIVER INFO
01:0011 CF                      BYTE    0xCF    ;SPECIFY COMMAND - FIRST PARAM
01:0012 02                      BYTE    0x02    ;SPECIFY COMMAND - SECOND PARA
01:0013 25                      BYTE    0x25    ;MOTOR OFF COUNTDOWN VALUE
                        ;
01:0014                 VARNFO:                 ;VARIABLE DISK DRIVER INFO
01:0014 00                      BYTE    0x00    ;SECTOR SIZE (2^N*128)
01:0015 00              VARSPT: BYTE    0x00    ;SECTORS PER TRACK (EACH SIDE)
01:0016 00                      BYTE    0x00    ;READ GAP LENGTH
01:0017 00                      BYTE    0x00    ;DTL
01:0018 00                      BYTE    0x00    ;FORMAT GAP LENGTH
                        ;
   0005                 VARLEN  == .-VARNFO     ;VARIABLE DISK DRIVER INFO LEN
                        ;
01:0019 E5                      BYTE    0xE5    ;FORMAT FILL BYTE
01:001A 19                      BYTE    0x19    ;HEAD SETTLE TIME (MILLISECONI
01:001B 04                      BYTE    0x04    ;MOTOR START TIME (1/8 SECONDS
                        ;
                        ;       1024 BYTE SECTOR, DOUBLE-DENSITY, TWO-SIDED (N
                        ;
01:001C 04              DSTBAS: BYTE    4       ;BLOCK SIZE
01:001D 00C8                    WORD    (40*(10*(1<<3)))/(1<<4)  ;NUMBER OF BL
01:001F 02                      BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
01:0020 03                      BYTE    3       ;PHYSICAL SECTOR SIZE (2^N*12
01:0021 000A                    WORD    10      ;PHYSICAL SECTORS PER TRACK
01:0023 0028                    WORD    40      ;PHYSICAL TRACKS PER DISK
01:0025 0000                    WORD    0       ;NUMBER OF RESERVED TRACKS
                        ;
   000B                 VAROFF  == .-DSTBAS     ;VARIABLE DISK DRIVER INFO OFT
```

```
                        ;
01:0027 03                      BYTE    0x03    ;SECTOR SIZE (2^N*128)
01:0028 05                      BYTE    0x05    ;SECTORS PER TRACK (EACH SIDE)
01:0029 35                      BYTE    0x35    ;READ GAP LENGTH
01:002A FF                      BYTE    0xFF    ;DTL
01:002B 74                      BYTE    0x74    ;FORMAT GAP LENGTH
                        ;
   0010                 DSTLEN  == .-DSTBAS     ;DST LENGTH
                        ;
                        ;       1024 BYTE SECTOR, DOUBLE-DENSITY, ONE-SIDED (MI
                        ;
01:002C 03                      BYTE    3       ;BLOCK SIZE
01:002D 00C8                    WORD    (40*(5*(1<<3)))/(1<<3)  ;NUMBER OF BLOC
01:002F 02                      BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
01:0030 03                      BYTE    3       ;PHYSICAL SECTOR SIZE (2^N*128)
01:0031 0005                    WORD    5       ;PHYSICAL SECTORS PER TRACK
01:0033 0028                    WORD    40      ;PHYSICAL TRACKS PER DISK
01:0035 0000                    WORD    0       ;NUMBER OF RESERVED TRACKS
01:0037 03                      BYTE    0x03    ;SECTOR SIZE (2^N*128)
01:0038 05                      BYTE    0x05    ;SECTORS PER TRACK (EACH SIDE)
01:0039 35                      BYTE    0x35    ;READ GAP LENGTH
01:003A FF                      BYTE    0xFF    ;DTL
01:003B 74                      BYTE    0x74    ;FORMAT GAP LENGTH
                        ;
                        ;       512 BYTE SECTOR, DOUBLE-DENSITY, TWO-SIDED (MI
                        ;
01:003C 04                      BYTE    4       ;BLOCK SIZE
01:003D 00A0                    WORD    (40*8*2*(1<<2))/(1<<4)  ;NUMBER OF BLO
01:003F 02                      BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
01:0040 02                      BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128
01:0041 0001                    WORD    1       ;PHYSICAL SECTORS PER TRACK
01:0043 0280                    WORD    40*8*2  ;PHYSICAL TRACKS PER DISK
01:0045 0001                    WORD    1       ;RESERVED TRACKS
01:0047 02                      BYTE    0x02    ;SECTOR SIZE (2^N*128)
01:0048 08                      BYTE    0x08    ;SECTORS PER TRACK (EACH SIDE)
01:0049 2A                      BYTE    0x2A    ;READ GAP LENGTH
01:004A FF                      BYTE    0xFF    ;DTL
01:004B 50                      BYTE    0x50    ;FORMAT GAP LENGTH
                        ;
                        ;       512 BYTE SECTOR, DOUBLE-DENSITY, ONE-SIDED (MI
                        ;
01:004C 03                      BYTE    3       ;BLOCK SIZE
01:004D 00A0                    WORD    (40*8*(1<<2))/(1<<3)  ;NUMBER OF BLOCK
01:004F 02                      BYTE    2       ;NUMBER OF DIRECTORY BLOCKS
01:0050 02                      BYTE    2       ;PHYSICAL SECTOR SIZE (2^N*128
01:0051 0001                    WORD    1       ;PHYSICAL SECTORS PER TRACK
01:0053 0140                    WORD    40*8    ;PHYSICAL TRACKS PER DISK
01:0055 0001                    WORD    1       ;RESERVED TRACKS
01:0057 02                      BYTE    0x02    ;SECTOR SIZE (2^N*128)
01:0058 08                      BYTE    0x08    ;SECTORS PER TRACK (EACH SIDE)
01:0059 2A                      BYTE    0x2A    ;READ GAP LENGTH
01:005A FF                      BYTE    0xFF    ;DTL
01:005B 50                      BYTE    0x50    ;FORMAT GAP LENGTH
                        ;
   0004                 NMBDST  == (.-DSTBAS)/DSTLEN  ;NUMBER OF DST'S
                        ;
```

```
)2:0000                         LOC     Code#   ;LOCATE IN CODE SEGMENT
                        ;
)2:0000 31C0            DSKIN_::XOR     AX,AX   ;GET INTERRUPT VECTOR SEGMENT
)2:0002 8EC0                    MOV     ES,AX   ;SET ES-REG
)2:0004 26 C706 0078            ES MOV  (0x1E*4)+0,WORD &DRVNFO  ;SET PARM OFFSI
        01:0011
)2:000B 26 8C0E 007A            ES MOV  (0x1E*4)+2,CS  ;SET PARM POINTER SEGMEN
)2:0010 B4 00                   MOV     AH,=0   ;SET FUNCTION NUMBER=0
)2:0012 CD 13                   INT     0x13    ;RESET DISK DRIVER
)2:0014 C3                      RET             ;DONE
                        ;
)2:0015 BB 01:000A      DSKDR_::MOV     BX,&DMXSPH  ;GET MUTUAL EXCLUSION SEMAP
02:0018 E8 03:0000*             CALL    WAIT#   ;DISPATCH IF NECESSARY
02:001B 8A54 01                 MOV     DL,PDRDRV[SI]  ;GET PD REQ DRIVE NUMBER
02:001E 8A1C                    MOV     BL,PDRFCN[SI]  ;GET PD REQ FUNCTION NUM
02:0020 80FB 05                 CMP     BL,=NMBFCN  ;VALID FUNCTION NUMBER?
02:0023 73 08                   JNC     __NVFN  ;IF NOT, CONTINUE
02:0025 B7 00                   MOV     BH,=0   ;MAKE FUNCTION NUMBER DOUBLE LE
02:0027 01DB                    ADD     BX,BX   ;X2
02:0029 FF97 01:0000            CALLI   FCNTBL[BX]  ;EXECUTE FUNCTION PROCESSOR
02:002D 50              __NVFN: PUSH    AX      ;SAVE RETURN CODE
02:002E BB 01:000A              MOV     BX,&DMXSPH  ;GET MUTUAL EXCLUSION SEMAP
02:0031 E8 04:0000*             CALL    SIGNAL# ;SIGNAL PROCESS AS READY
02:0034 58                      POP     AX      ;RESTORE RETURN CODE
02:0035 C3                      RET             ;DONE
                        ;
02:0036 E8 0094         RDDSK:  CALL    SETUP   ;DO COMMON SETUP
02:0039 B4 02                   MOV     AH,=2   ;SET FUNCTION NUMBER=2
02:003B EB 75                   JMPS    RWFCOM  ;CONTINUE
                        ;
02:003D E8 008D         WRDSK:  CALL    SETUP   ;DO COMMON SETUP
02:0040 B4 03                   MOV     AH,=3   ;SET FUNCTION NUMBER=3
02:0042 EB 6E                   JMPS    RWFCOM  ;CONTINUE
                        ;
02:0044 BB 01:001C      FMTDSK: MOV     BX,&DSTBAS  ;GET DST BASE ADDRESS
02:0047 8B44 0E                 MOV     AX,PDRDST[SI]  ;GET PDR DST ADDRESS
02:004A 85C0                    TEST    AX,AX   ;PDR DST ADDRESS=0?
02:004C 75 03                   JNZ     __TDSF  ;IF NOT, CONTINUE
02:004E 83C3 20                 ADD     BX,=DSTLEN*2  ;ELSE, ADVANCE TO NEXT DS
02:0051 E8 00BF         __TDSF: CALL    SETNFO  ;SET DISK DRIVER INFO
02:0054 8A6C 02                 MOV     CH,BYTE PDRTRK[SI]  ;GET REQUESTED TRAC
02:0057 84ED                    TEST    CH,CH   ;REQUESTED TRACK NUMBER=0?
02:0059 75 04                   JNZ     __NTK0  ;IF NOT, CONTINUE
02:005B B4 00                   MOV     AH,=0   ;ELSE, SET FUNCTION NUMBER=0
02:005D CD 13                   INT     0x13    ;RESET DISK DRIVER
02:005F B6 00           __NTK0: MOV     DH,=0   ;PRESET HEAD NUMBER=0
02:0061 F644 05 80              TEST    PDRSEC+1[SI],=BYTE 1<<7  ;HEAD ONE BIT
02:0065 74 02                   JZ      __NH1   ;IF NOT, CONTINUE
02:0067 FEC6                    INC     DH      ;ELSE, SET HEAD NUMBER=1
02:0069 8B5C 0A         __NH1:  MOV     BX,PDRDMA[SI]  ;GET REQUESTED DMA OFFSE
02:006C 8E44 0C                 MOV     ES,PDRBAS[SI]  ;GET REQUESTED DMA BASE
02:006F B8 0501                 MOV     AX,=0x0501  ;SET FUNCTION=5/SECTOR COUI
02:0072 EB 3E                   JMPS    RWFCOM  ;CONTINUE
                        ;
02:0074 30C0            RETDST: XOR     AL,AL   ;PRESET RETURN CODE=0
02:0076 80FA 04                 CMP     DL,=4   ;VALID DRIVE NUMBER?
```

```
02:0079 73 33                    JNC     __X     ;IF INVALID DRIVE, CONTINUE
02:007B 88C6                     MOV     DH,AL   ;ELSE, SET HEAD NUMBER=0
02:007D B9 0001                  MOV     CX,=1   ;SET TRACK NUMBER=0/SECTOR NUMBI
02:0080 BB 01:001C               MOV     BX,&DSTBAS  ;GET DST BASE ADDRESS
02:0083 BD 0002                  MOV     BP,=NMBDST/2  ;GET NUMBER OF DST'S (/2)
02:0086 E8 008A          __DSTL: CALL    SETNFO  ;SET DISK DRIVER INFO
02:0089 B8 0401                  MOV     AX,=0x0401  ;SET FUNCTION=4/SECTOR COUN'
02:008C E8 0023                  CALL    RWFCOM  ;ATTEMPT TO READ TRACK 0/SECTOR
02:008F FEC0                     INC     AL      ;REQUESTED DRIVE READY?
02:0091 75 07                    JNZ     __RDY   ;IF SO, CONTINUE
02:0093 83C3 20                  ADD     BX,=DSTLEN*2  ;ELSE, ADVANCE TO NEXT DS'
02:0096 4D                       DEC     BP      ;DECREMENT NUMBER OF DST'S
02:0097 75 ED                    JNZ     __DSTL  ;IF NOT LAST DST, CONTINUE
02:0099 C3                       RET             ;ELSE, DONE
02:009A FEC6             __RDY:  INC     DH      ;SET HEAD NUMBER=1
02:009C B8 0401                  MOV     AX,=0x0401  ;SET FUNCTION=4/SECTOR COUN
02:009F E8 0010                  CALL    RWFCOM  ;ATTEMPT TO READ TRACK 0/SECTOR
02:00A2 84C0                     TEST    AL,AL   ;READ SUCCESSFUL?
02:00A4 74 03                    JZ      __RDST  ;IF SO, CONTINUE
02:00A6 83C3 10                  ADD     BX,=DSTLEN  ;CALC ONE-SIDED DST ADDRESS
02:00A9 895C 0E          __RDST: MOV     PDRDST[SI],BX  ;SET DST ADDRESS
02:00AC B0 FF                    MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
02:00AE C3               __X:    RET             ;DONE
                         ;
02:00AF B0 FF            RETRDY: MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
02:00B1 C3                       RET             ;DONE
                         ;
02:00B2 C606 01:0010 04  RWFCOM: MOV     TRYCNT,=MAXTRY  ;SET TRY COUNT=MAX TRYS
02:00B7 50               __RDL:  PUSH    AX      ;SAVE FUNCTION/SECTOR COUNT
02:00B8 CD 13                    INT     0x13    ;READ/WRITE/VERIFY SECTOR
02:00BA 58                       POP     AX      ;RESTORE FUNCTION/SECTOR COUNT
02:00BB 73 0D                    JNC     __X     ;IF NO ERRORS, CONTINUE
02:00BD FE0E 01:0010             DEC     TRYCNT  ;ELSE, DECREMENT TRY COUNT
02:00C1 75 F4                    JNZ     __RDL   ;IF NOT LAST TRY, CONTINUE
02:00C3 B4 00                    MOV     AH,=0   ;ELSE, SET FUNCTION NUMBER=0
02:00C5 CD 13                    INT     0x13    ;RESET DISK DRIVER
02:00C7 B0 FF                    MOV     AL,=0xFF  ;SET RETURN CODE=0FFH
02:00C9 C3                       RET             ;DONE
02:00CA 30C0             __X:    XOR     AL,AL   ;SET RETURN CODE=0
02:00CC C3                       RET             ;DONE
                         ;
02:00CD 8A6C 02          SETUP:  MOV     CH,BYTE PDRTRK[SI]  ;GET TRACK NUMBER
02:00D0 8A4C 04                  MOV     CL,BYTE PDRSEC[SI]  ;GET SECTOR NUMBER
02:00D3 807C 14 03               CMP     SECSIZ[SI],=3  ;PHYSICAL SECTOR SIZE=3?
02:00D7 74 1B                    JZ      __PSS3  ;IF SO, CONTINUE
02:00D9 8B44 02                  MOV     AX,PDRTRK[SI]  ;GET REQUESTED TRACK NUM
02:00DC 8A4C 10                  MOV     CL,BLKSIZ[SI]  ;GET ALLOCATION BLOCK SI
02:00DF D3E8                     SHR     AX,CL   ;CALC TRACK NUMBER
02:00E1 88C5                     MOV     CH,AL   ;TRACK NUMBER TO CH-REG
02:00E3 8A4C 10                  MOV     CL,BLKSIZ[SI]  ;GET ALLOCATION BLOCK SI
02:00E6 F6D9                     NEG     CL      ;NEGATE ALLOCATION BLOCK SIZE
02:00E8 80C1 08                  ADD     CL,=8   ;CALC SHIFT COUNT
02:00EB B0 FF                    MOV     AL,=0xFF  ;SET AL=0FFH
02:00ED D2E8                     SHR     AL,CL   ;CALC ALLOCATION BLOCK MASK
02:00EF 8A4C 02                  MOV     CL,BYTE PDRTRK[SI]  ;GET REQUESTED TRAC
02:00F2 20C1                     AND     CL,AL   ;CALC SECTOR NUMBER
```

```
02:00F4 8B5C 0E          __PSS3: MOV     BX,PDRDST[SI]  ;GET PDR DST ADDRESS
02:00F7 E8 0019                  CALL    SETNFO  ;SET DISK DRIVER INFO
02:00FA B6 00                    MOV     DH,=0   ;PRESET HEAD NUMBER=0
02:00FC A0 01:0015               MOV     AL,VARSPT  ;GET VARIABLE INFO SECTORS
02:00FF 38C1                     CMP     CL,AL   ;SECTOR ON SIDE ONE?
02:0101 72 04                    JC      __SOS0  ;IF NOT, CONTINUE
02:0103 28C1                     SUB     CL,AL   ;ELSE, CALC SECTOR NUMBER
02:0105 FEC6                     INC     DH      ;SET HEAD NUMBER=1
02:0107 FEC1             __SOS0: INC     CL      ;INCREMENT SECTOR NUMBER
02:0109 8B5C 0A                  MOV     BX,PDRDMA[SI]  ;GET REQUESTED DMA OFF
02:010C 8E44 0C                  MOV     ES,PDRBAS[SI]  ;GET REQUESTED DMA BAS
02:010F 8A44 06                  MOV     AL,BYTE PDRSC[SI]  ;GET PD REQ SECTOR
02:0112 C3                       RET             ;DONE
                         ;
02:0113 53               SETNFO: PUSH    BX      ;SAVE CX-REG
02:0114 51                       PUSH    CX      ;SAVE CX-REG
02:0115 56                       PUSH    SI      ;SAVE SI-REG
02:0116 57                       PUSH    DI      ;SAVE DI-REG
02:0117 83C3 0B                  ADD     BX,=VAROFF  ;ADD VARIABLE DRIVER INFO
02:011A 89DE                     MOV     SI,BX   ;VARIABLE DISK DRIVER INFO TO
02:011C BF 01:0014               MOV     DI,&VARNFO  ;GET VARIABLE INFO ADDRES:
02:011F 8CD9                     MOV     CX,DS   ;GET DS-REG
02:0121 8EC1                     MOV     ES,CX   ;SET ES-REG
02:0123 B9 0005                  MOV     CX,=VARLEN  ;GET VARIABLE INFO LENGTH
02:0126 FC                       CLD             ;CLEAR DIRECTION FLAG
02:0127 F3 A4                    REP MOVS BYTE   ;SET VARIABLE DISK DRIVER INF(
02:0129 5F                       POP     DI      ;RESTORE DI-REG
02:012A 5E                       POP     SI      ;RESTORE SI-REG
02:012B 59                       POP     CX      ;RESTORE CX-REG
02:012C 5B                       POP     BX      ;RESTORE CX-REG
02:012D C3                       RET             ;DONE
                         ;
   0000                  END
```

```
                        ;
                        ; VERSION: 01/03/84
                        ;
   0000                         MODULE  "MSTIPC"  ;MODULE NAME
                        ;
01:0000                         LOC     Data#     ;LOCATE IN DATA SEGMENT
                        ;
01:0000 01              MEMTBL::BYTE    1         ;NUMBER OF MEMORY SEGMENTS
01:0001 0050                    WORD    0x0050    ;MEMORY SEGMENT BASE
01:0003 3FB0                    WORD    0x4000-0x0050  ;MEMORY SEGMENT LENGTH
                        ;
   0000                 END
```