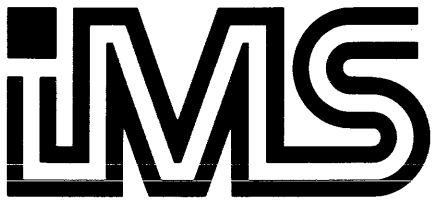


INTERNATIONAL

DIVISION OF L/F TECHNOLOGIES INC.

TURBODOS 1.4

Z80 IMPLEMENTOR'S GUIDE

TurboDOS 1.4

Z80 Implementor's Guide

June 1984

Copyright 1984

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

All rights reserved.

TurboDOS[R] is a registered trademark of Software 2000, Inc.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Copyright Notice

Copyright 1984 by Software 2000, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Software 2000, Inc., 1127 Hetrick Avenue, Arroyo Grande, California 93420, U.S.A.

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.

CP/M, CP/M Plus, 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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

ABOUT THIS GUIDE

Purpose

We've designed this Z80 Implementor's Guide to provide the information you need to know in order to generate various TurboDOS configurations for Z80-based 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 Z80-based microcomputer hardware and assembly-language programming. We've also assumed you have read both the User's Guide and the Z80 Programmer's Guide, and are therefore familiar with the commands, external features, and internal functions of Z80 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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 Z80 Programmer's Guide
- TurboDOS 1.4 8086 Programmer's Guide
- TurboDOS 1.4 8086 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 Z80 Programmer's Guide explains the internal workings of Z80 TurboDOS, and describes each operating system function in detail.

You'll need the 8086 guides if you are programming or configuring a TurboDOS system that uses 8086-family microprocessors.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

ARCHITECTURE

Module Hierarchy 1-1
Process Level 1-1
Kernel Level 1-2
Driver Level 1-2
TurboDOS Loader 1-2
Module Flow Diagram 1-3
Process Modules 1-4
Kernel Modules 1-5
Driver Modules 1-8
Standard Packages 1-8
Package Contents Table 1-9
Additional Modules 1-10
Memory Required 1-11
Other Languages 1-12

SYSTEM GENERATION

Introduction 2-1
GEN Command 2-2
Patch Points 2-7
Network Operation 2-20
Network Model 2-20
Network Tables 2-20
Message Forwarding 2-22
A Complex Example 2-24
Sysgen Procedure 2-26

DISTRIBUTION

TurboDOS Licensing 3-1
Legal Protection 3-1
User Obligations 3-2
Dealer Obligations 3-2
Distributor Obligations 3-3
Serialization 3-4
Technical Support 3-5
SERIAL Command 3-6
PACKAGE Command 3-8
Distribution Procedure 3-10

Copyright 1984 by Software 2000, Inc.
All rights reserved.

CODING CONVENTIONS	Assembler Notes	4-1
	Undefined External References	4-2
	Memory Allocation	4-3
	List Processing	4-4
	Task Dispatching	4-5
	Interrupt Service	4-7
	Poll Routines	4-8
	Mutual Exclusion	4-9
	Sample Driver Using Interrupts	4-10
	Sample Driver Using Polling	4-11
	Special Segments	4-12
	?INIT? Segment	4-12
	?PAGE? Segment	4-12
	?BANK? Segment	4-12
	Inter-Process Messages	4-13
	Console Routines	4-14
	Sign-On Message	4-14
	Resident Process	4-15
	User-Defined Function	4-16
DRIVER INTERFACE	General Notes	5-1
	Initialization	5-2
	Console Driver	5-3
	Printer Driver	5-5
	Disk Driver	5-6
	Bank-Select Driver	5-9
	Network Driver	5-10
	Comm Driver	5-14
	Clock Driver	5-15
	Bootstrap	5-17
APPENDIX	Sample Driver Source Listings	A-1

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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. Drivers are also required for the real-time clock (or other periodic interrupt source), and for bank-switched memory (if applicable).

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

TurboDOS Loader

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Process Modules

Module	Function
LCLUSR	Responsible for supporting local user's TPA activities. Not Disk operations.
LCLMSG	Contains all O/S error messages.
LCLTBL	Local user option table. To hold PAR patches.
CMDINT	Command interpreter, processes commands from local user. From Line editor. This RTN does parsing.
AUTLOD	Autoload routine which processes COLDSTRT.AUT and WARMSTRT.AUT. this is in STD, but is Technically optional.
SGLUSR	Routine to flush/free disk buffers at each console input. Use for single-user configurations instead of FLUSHR.
AUTLOG (OPTIONAL)	Automatic log-on routine. Used when full log-on security is not desired. See AUTUSR patch point.
SUBMIT	Routine to emulate CP/M processing of $$$.SUB files. (Use is not recommended.) Do not use!! 20% speed degredation.
NETSVC	Services network requests from other processors on the network. NETREQ - Sends messages.
NETTBL	Tables to define local network topology, used by NETSVC+NETREQ.
NETFWD	Manages network message forwarding. Requires NETREQ+NETSVC.
MSGFMT	- Formats + structures messages.
DSPOOL	Processes background printing.
FLUSHR	Periodically flushes disk buffers. Use for network master configuration instead of SGLUSR.

Provide network interaction.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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	Support routines for FILMGR.
FILCOM	Processes common file requests always processed locally.
FILLOK	File- and record-level interlock routines called by FILMGR.
FFOMGR	FIFO support, called by FILLOK.
DRVLOK	Drive interlock routines.
FASLOD	Program loader incorporating an optimizer for fastest loading.
NORLOD	Unoptimized program loader, an alternative to FASLOD.
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 and FASLOD.
DSKTBL	Table defining drives A-P as local or remote disk drives.
NONFIL	Processes non-file functions.
COMMGR	Processes comm-channel funct's.

USE ONE OF OTHER.

USE NORLOAD ONLY FOR MP/M QUEUES.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Kernel Modules (Continued)

Module	Function
CPMSUP	Processes C-functions 7, 8, 24, 28, 29, 31, 37, 107 (optional).
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.
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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Kernel Modules
(Continued)

Module	Function
RTCMGR	Real-time clock manager keeps system date and time.
NETLOD	Loads programs over the network. USED IN SLAVE
BNKMGR	Responsible for bank-switching in banked-memory systems.
BNKREQ	Alternative to NETLOD for use in banked-memory systems. Superset of NETLOD
DSPCHR	Multi-task dispatcher which controls sharing of the local processor among multiple processes. Tells O/S # of process that is up now. Assign priority.
DSPSGL	Null dispatcher used as alternative to DSPCHR when only one process is required (OSLOAD.COM and single-user w/o spooling).
MEMMGR	Memory manager responsible for dynamic allocation of memory.
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~~ 256 bytes of zeroes, may be included to provide patch area. Varies

one or other

WORDSTAR NEEDS A REAL-TIME CLOCK DRIVER. RTCNUL IS NO GOOD FOR THIS.

PATCH SOURCE → NAME ('PATCH')
Patch:RES 0200H ; reserve re-
END
... with M80

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.
SELBNK	Bank-select driver for banked-memory systems.
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 facing page. Most TurboDOS requirements can be satisfied by linking the appropriate standard package together with a few additional optional modules plus the requisite driver modules.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Listing of contents of STANDARD PACKAGES.

Module	K	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
AUTLOD	.2	-	AUTLOD	AUTLOD	AUTLOD	AUTLOD	AUTLOD
AUTLOG	.0	-	AUTLOG	AUTLOG	AUTLOG	AUTLOG	AUTLOG
BNKMGR	2.0	-	+	+	+	+	+
BNKREQ	.3	-	-	-	+	+	+
BUFMGR	1.1	BUFMGR	BUFMGR	BUFMGR	BUFMGR	-	BUFMGR
CMDINT	1.3	-	CMDINT	CMDINT	CMDINT	CMDINT	CMDINT
COMMGR	.1	-	COMMGR	COMMGR	COMMGR	COMMGR	COMMGR
COMSUB	.3	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB
CONMGR	.3	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR
CONREM	.4	-	-	-	+	-	-
CONTBL	.0	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL
CPMSUP	.2	-	+	+	+	+	+
DOMGR	.4	-	DOMGR	DOMGR	DOMGR	DOMGR	DOMGR
DRVLOK	.2	-	-	-	DRVLOK	-	-
DSKMGR	.6	DSKMGR	DSKMGR	DSKMGR	DSKMGR	-	DSKMGR
DSKTBL	.0	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL
DSPCHR	.7	-	-	DSPCHR	DSPCHR	DSPCHR	DSPCHR
DSPOOL	.9	-	-	DSPOOL	DSPOOL	-	DSPOOL
DSPSGL	.2	DSPSGL	DSPSGL	-	-	-	-
FASLOD	.4	-	+	+	+	+	+
FFOMGR	.9	-	-	-	FFOMGR	-	-
FILCOM	.4	FILCOM	FILCOM	FILCOM	FILCOM	FILCOM	FILCOM
FILLOK	1.7	-	-	-	FILLOK	-	-
FILMGR	2.1	FILMGR	FILMGR	FILMGR	FILMGR	-	FILMGR
FILSUP	2.4	FILSUP	FILSUP	FILSUP	FILSUP	-	FILSUP
FLUSHR	.2	-	-	-	FLUSHR	-	-
INPLN	.1	-	INPLN	INPLN	INPLN	INPLN	INPLN
LCLMSG	.4	-	LCLMSG	LCLMSG	LCLMSG	LCLMSG	LCLMSG
LCLTBL	.0	-	LCLTBL	LCLTBL	LCLTBL	LCLTBL	LCLTBL
LCLUSR	1.2	-	LCLUSR	LCLUSR	LCLUSR	LCLUSR	LCLUSR
LDRMSG	.2	LDRMSG	-	-	-	-	-
LSTMGR	.2	-	LSTMGR	LSTMGR	LSTMGR	LSTMGR	LSTMGR
LSTTBL	.1	-	LSTTBL	LSTTBL	LSTTBL	LSTTBL	LSTTBL
MEMMGR	.3	-	MEMMGR	MEMMGR	MEMMGR	MEMMGR	MEMMGR
MPMSUP	.1	-	+	+	+	+	+
MSGFMT	.1	-	-	-	+	MSGFMT	MSGFMT
NETFWD	.3	-	-	-	+	+	+
NETLOD	.4	-	-	-	+	+	+
NETMGR	.9	-	-	-	NETMGR	NETMGR	NETMGR
NETREQ	1.5	-	-	-	+	NETREQ	NETREQ
NETSVC	1.7	-	-	-	NETSVC	+	+
NETTBL	.0	-	-	-	NETTBL	NETTBL	NETTBL

the + means that this is sometimes included in.
— not normally included.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Module	K	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
NONFIL	.2	NONFIL	NONFIL	NONFIL	NONFIL	NONFIL	NONFIL
NORLOD	.1	-	+	+	+	+	+
OSLOAD	1.3	OSLOAD	-	-	-	-	-
OSNTRY	.5	OSNTRY	OSNTRY	OSNTRY	OSNTRY	OSNTRY	OSNTRY
PATCH	.1	+	+	+	+	+	+
QUEMGR	1.1	-	-	-	+	+	+
RTCMGR	.1	-	RTCMGR	RTCMGR	RTCMGR	-	RTCMGR
RTCNUL	.1	+	+	+	+	+	+
SGLUSR	.1	-	SGLUSR	SGLUSR	-	-	SGLUSR
SPLMSG	.1	-	-	SPLMSG	SPLMSG	SPLMSG	SPLMSG
SPOOLR	.5	-	-	SPOOLR	SPOOLR	SPOOLR	SPOOLR
SUBMIT	.1	-	+	+	+	+	+
SYSNIT	.0	-	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
BNKMGR	All systems with banked memory.
BNKREQ	Banked systems that load programs over the network.
CONREM	Network masters with no console (instead of CONDR@).
CPMSUP	To support C-fcns 7, 8, 24, 28, 29, 31, 37 and 107.
FASLOD	Non-banked systems that load pgms from local disks.
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	Non-banked systems that load pgms over the network.
NETREQ	Network masters that make requests over the network.
NORLOD	Smaller, unoptimized alternative to FASLOD (above).
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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 on non-banked hardware:

	LOADR	SINGL	SPOOL	MASTR	SLAVE	SLAVX
O/S	10K	13K	15K	20K	10K	18K
Drivers	2K	2K	2K	3K	1K	2K
Buffers	4K	4K	4K	16K	-	4K
Dynamic	1K	1K	1K	3K	2K	2K
Total	17K	20K	22K	42K	13K	26K
TPA	-	44K	42K	22K	51K	38K

In banked-memory systems, a full 63K TPA is always available.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Other Languages

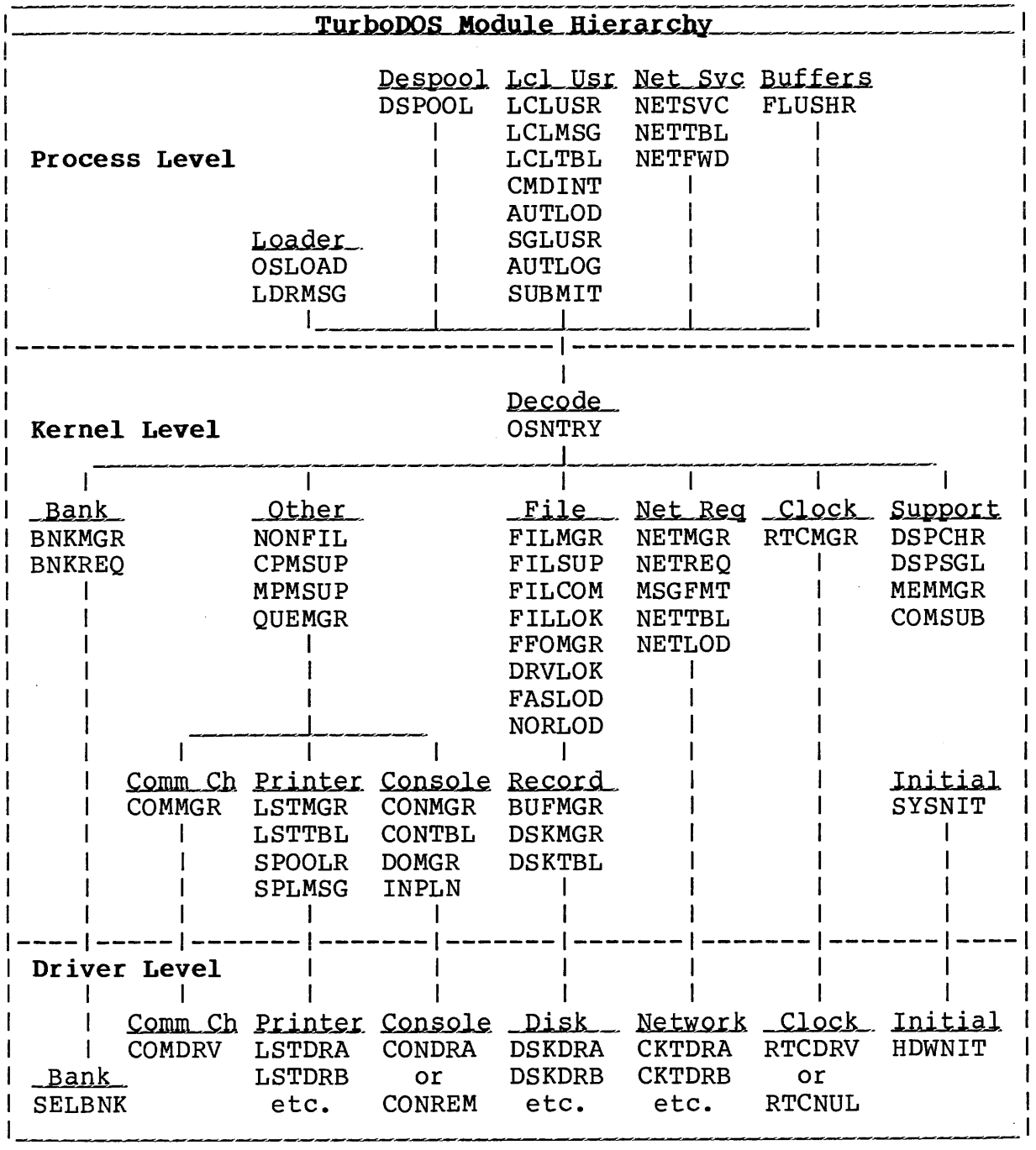
To facilitate translation into languages other than English, TurboDOS has been implemented with all textual messages segregated into separate modules. All such message modules are available in source form to TurboDOS OEM licensees upon request.

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.COM.

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.



Copyright 1984 by Software 2000, Inc.
All rights reserved.

SYSTEM GENERATION This section explains the TurboDOS system generation procedure in detail. It describes how to use the GEN command 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 form (.REL files). Hardware-dependent driver modules are furnished in the same fashion. The TurboDOS GEN command is a specialized linker used to bind the desired combination of modules together into an executable version of TurboDOS. The GEN command 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 the GEN command several times. At minimum, you have to generate both a loader OSLOAD.COM 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 GEN to generate a cold-start bootstrap routine for the start-up PROM or boot track.

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

GEN Command

The GEN command is a specialized linker for software modules in Microsoft relocatable format, and is designed primarily for use in TurboDOS system generation.

Syntax

```
GEN srcefile {destfile} {;options}
```

Explanation

The GEN command links a specified collection of relocatable modules together into a single executable program. The "srcefile" argument specifies the names of two input files: a configuration file "srcefile.GEN" and a parameter file "srcefile.PAR". The "destfile" argument specifies the name of the executable output file to be created (normally type .COM or .SYS). If "destfile" is omitted, then the "srcefile" argument is also used as the name of the executable output file, and should include an explicit file type (.COM or .SYS).

If the configuration file "srcefile.GEN" is found, it must contain the list of relocatable modules (.REL files) to be linked together. If the configuration file is not found, then the GEN 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:

```
relfile {,relfile}... {;comment}
```

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

After obtaining the list of modules from the file or console, GEN links all of the modules together, a two-pass process that displays the name of each module as it is encountered.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Explanation (Continued)

When the linking phase is complete, GEN looks for a parameter file "srcefile.PAR" and processes it if found. The parameter file (if present) must be a text file containing symbolic patches. 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, a hexadecimal number, or an expression composed of names and hex numbers connected by + or - operators. Hex numbers must begin with a digit (for example, 0FFFF) to distinguish them from names. The "location" expression must be followed by an equal-sign = character.

The "value" arguments may be expressions (as defined above) or quoted ASCII strings, and must be separated by commas. A "value" expression is stored as a 16-bit word if its value exceeds 255 or if it is enclosed in parentheses; otherwise, it is stored as an 8-bit byte. A quoted ASCII string may be enclosed by either quotes "..." or apostrophes '...', and is stored as a sequence of 8-bit bytes. Within a quoted string, ASCII control characters may be specified by using circumflex (example: "^X" denotes CTRL-X).

After the .PAR file (if any) is processed and the necessary patches made, GEN writes the executable file out to disk.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Explanation (Continued)

Each relocatable TurboDOS module 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. The GEN command verifies that all modules to be linked are serialized consistently, and serializes the executable file accordingly.

Options

Option	Explanation
;Kxxxx	Indicates that a system for a banked-memory environment is to be generated, and defines the hexadecimal base address "xxxx" of the common (non-switched) memory segment.
;Lxxxx	Defines the hexadecimal address "xxxx" as the lower boundary of the executable program. Default for .COM files is ;L0100.
;M	Prints a load map.
;S	Prints a sorted symbol table.
;Uxxxx	Defines the hexadecimal address "xxxx" as the upper boundary of the executable program. Default for .SYS files is ;UFFFF.
;X	Diagnoses any references to undefined symbols. Default is not to diagnose such references, since they are quite normal in TurboDOS system generation.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Example

In the following example, GEN is used to link a single-user TurboDOS system for a banked-memory system, using the modules listed in BNKSINGL.GEN and the patches in BNKSINGL.PAR, creating the executable file BNKSINGL.SYS.

```
0A}GEN BNKSINGL.SYS ;MKC000
Copyright 1984, Software 2000, Inc.
CNFTBL   ;CONFIGURATION TABLE
STDSINGL ;STANDARD SINGLE-USER SYSTEM
BNKMGR   ;BANK MEMORY MANAGER
CPMSUP   ;CP/M FUNCTION SUPPORT
USRSOM   ;USER SIGN-ON MESSAGE AREA
NITIMS   ;HARDWARE INITIALIZATION
MPENIT   ;MEMORY PARITY
CON96    ;ASCII CONSOLE AT 9600 BAUD
LSTCTS   ;CLEAR TO SEND SERIAL DRIVER
LSTXON   ;X-ON,X-OFF SERIAL DRIVER
etc.

Pass 1
CONFIG LCLUSR LCLMSG LCLTBL CMDINT etc.

Pass 2
CONFIG LCLUSR LCLMSG LCLTBL CMDINT etc.

Processing parameter file:
USRSOM = 0D,0A,"IMS International,
          TurboDOS-8 1.4 (Bank Single)$"
AUTUSR = 80 ;LOGON TO USER 0, PRIVILEDGED
COMPAT = 0B0;COMPATIBILITY FLAGS
PRTMOD = 0  ;DEFAULT TO PRINT DIRECT
NMBUFS = 2  ;NUMBER BUFFERS (2)
SRT401 = 6  ;6 MS. STEP RATE FOR 8"
            ;DRIVES ON 401 CONTROLLER
SRT431 = 6  ;12 MS. STEP RATE FOR 5"
            ;DRIVES ON 431 CONTROLLER
etc.

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Error Messages

File name missing from command
Invalid input file name
Non-privileged user
Serial number violation
Not enough memory
Vacuous input file(s)
Unexpected EOF in input file
Disk is full
Can't make output file
No input files
Can't open input file
Load address out-of-bounds
Multiple defined starting address
Duplicate symbol: <name>
Undefined symbol: <name>

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points

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

Symbol	Default Value	Module
ABTCHR	"^C"	CONTBL
	Abort character (after attention).	
ATNBEL	"^G"	CONTBL
	Attention-received warning character.	
ATNCHR	"^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	0FF	AUTLOG
	Automatic log-on user number. Default value of 0FF requires that user log-on via LOGON command. If automatic log-on desired at cold-start, patch AUTUSR to the desired user number (00-1F), and set the sign-bit if a privileged log-on is desired. Generally patched to 80 in single-user systems to cause automatic privileged log-on to user zero.	

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points
(Continued)

Symbol	Default Value	Module
BFLDLY = (012C)		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). You can make it faster or slower, But do not make it Ø		
BUFSIZ = 3		BUFMGR
Default disk buffer size (0=128, 1=256, 2=512, 3=1K,..., 7=16K). Default value specifies 1K disk buffers. Normally using 05 on 1.4 MASTER.		
CKTAST = (0000),CKTDRA, (0100),CKTDRB, (0200),CKTDRC, (0300),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 = 9D up to 255.		CMDINT
Command line buffer length defines longest permissible command line. The default value permits two 80-char lines.		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
CLPCHR = "}"		CMDINT
Command line prompt character.		
CLSCHR = "\"		CMDINT
Command line separator character.		
COLDFN = 0,"COLDSTRT","AUT"		AUTLOD
File name and drive for cold-start auto-load processing (in FCB format).		
COMPAT = 0		FILCOM
Default compatibility flags which define rules to be used for file-sharing. Patch to 0F8 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 = 31		NONFIL
CP/M BDOS version number returned by C-function 12 in L-register.		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
CURBNK = 1		BNKMGR
Initial memory bank selected for TPA at cold-start. Applicable to banked-memory systems only. Patch to 0 to select non-banked mode at cold-start.		
DEFDID = (0000)		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.		
DSKAST =	00,DSKDRA,01,DSKDRB, 0FF,(0000),0FF,(0000),...	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.		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
DSKAST (Continued)		DSKTBL
For an invalid drive, the first byte must be 0FF, and the following word should be (0000).		
NOTE: In slave configurations STDSLAVE and STDSLAVX, the default values are:		
DSKAST = 80,(0000),81,(0000), 82,(0000),83,(0000), ...,8E,(0000),8F,(0000)		
DSPPAT = 01,01,01,...,01		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. Hex values 01 through 10 correspond to queues A-P, and 0 means that the printer is off-line. The default value assigns all printers to queue A. position represents the printer, the value represents the queue.		
ECOCHR = "^P"		CONTBL
Echo-print character (after attention). modify to put a message or status line.		
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. Recommend that you use a 12 (1A hex)		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points
(Continued)

Symbol	Default Value	Module
FWDTBL = ~~(0FFFF),(0FFFF),(0FFFE),~~ ~~(0FFFF),0FF~~ 0xFF, 0xFF 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 0FF, and defaults to empty.		
LDCOLD = 0FF		AUTLOD
Cold-start autoload enable flag. Patch to zero if you want to disable the cold-start autoload feature (COLDSTRT.AUT).		
LDWARM = 0FF		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.		

IN a network, each Box should have a different system name)

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points
(Continued)

Symbol	Default Value	Module
LOGUSR = 1F		FILCOM
User number for logged-off state. Default value is 31 decimal.		
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.		
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.		
MEMBLL = (1100)	Bottom address of O/S in absolute terms.	MEMMGR
Memory base lower limit, prevents allocation of dynamic memory space below this address when bank 0 is selected. Default value guarantees minimum of 4K TPA in bank 0 (enough to run BANK or BUFFERS).		

Do not Define As A Rule

Let the System do it!

NEVER PATCH THIS

It is just Large enough to run Buffers.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
MEMRES = (0100)		LCLUSR
Memory reserve, used when loading a program into TPA to provide a safety margin between the base of dynamic memory space and the top of bank 0 TPA. This allows dynamic space to grow by MEMRES bytes before the program in bank 0 TPA has to be aborted by TurboDOS. The MEMRES value may have to be increased above the 256-byte default value for reliable operation especially in non-banked network masters.		
MEMTOP = (0FFFF)		OSLOAD
Top of memory address for purposes of the RAM diagnostic test performed by OSLOAD. Patch to (0000) to omit test altogether.		
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 obtaining more TPA by reducing the size of the disk buffer pool. 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).		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
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 obtaining more TPA by reducing the size of the disk buffer pool. 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.)		
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).		
PRTCHR = "^L"		CONTBL
End-print character (after attention). This is a console attention-response, not to be confused with EOPCHR.		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
PRTMOD = 1		LCLTBL
Initial print mode for local user. The default value of 1 specifies spooling. Patch to 0 for direct, or 2 for console.		
PTRAST = 00,LSTDRA,0FF,(0000), 0FF,(0000),0FF,(0000),...		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 to be used. 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. ↘ usually the master or other network. For an invalid printer, the entry should be 0FF,(0000). NOTE: In slave configurations STDSLAVE and STDSLAVX, the default values are: PTRAST = 80,(0000),81,(0000), 82,(0000),83,(0000), ...,8E,(0000),8F,(0000)		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
QUEAST =	00,(0000),0FF,(0000), 0FF,(0000),0FF,(0000),...	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 entry should be 0FF,(0000). NOTE: In slave configurations STDSLAVE and STDSLAVX, the default values are: QUEAST = 80,(0000),81,(0000), 82,(0000),83,(0000), ...,8E,(0000),8F,(0000)		
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.		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
QUEDRV = 0FF		QUEMGR
Drive used for FIFOs that emulate MP/M queues. Default value 0FF means use the system disk (disk from which TurboDOS was loaded at cold-start). Patch to 00 - 0F 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, hex values 01 through 10 correspond to letters A-P, and zero means do not queue or print off-line.		
RCNMSK = 0FF		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 = "^Q"		CONTBL
Resume character (after attention).		

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Patch Points (Continued)

Symbol	Default Value	Module
SCANDN = 0		OSLOAD
Scan direction flag for OSLOAD. Patch to 0FFH 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. must reside with OS master.		
SPLDRV = 0FF		LCLTBL
Initial spool drive. Default value 0FF spools to system disk (from which TurboDOS was loaded at cold-start). Patch to 00 - 0F to specify a drive A-P.		
SRHDRV = 0		CMDINT
Search drive for command files. Patch to 01 - 10 hex to search drive A-P if command is not found on current drive, or patch t0 0FF to search system disk (from which TurboDOS was loaded at cold-start). Default value 0 disables this feature.		
SUBFN = 0,"$$$ ","SUB"		SUBMIT
Submit file name searched for by optional CP/M submit-file emulator.		
WARMFN = 0,"WARMSTRT","AUT"		AUTLOD
File name and drive for warm-start auto-load processing (in FCB format).		

change to OFF this will allow floppy priority for Loading.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

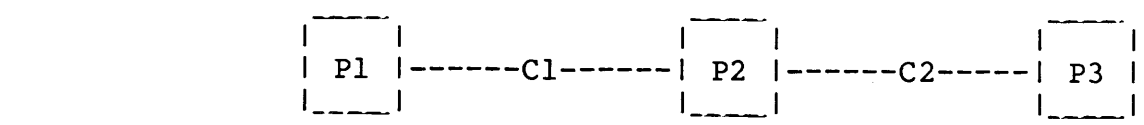
These tables are pre-defined with default values to make set-up of simple master/slave networks very easy. For complex multi-circuit networks, the set-up is somewhat more complicated (as might be expected).

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Message Forwarding

The forwarding module of TurboDOS (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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 an S-100 microcomputer system consisting of a Z80 CPU board, a 64K memory board, hard disk and floppy disk controller boards (all these make up the master processor), and several single-board slave processors on the same bus. The master processor is interfaced to two printers, one daisywheel and the other matrix, via RS232 serial ports. The daisywheel printer is on serial port 0 and uses XON/XOFF protocol, while the matrix printer is on port 1 and uses clear-to-send handshaking. In addition, the master has a high-speed RS422 interface connecting it to another S-100 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 S-100 system to access the hard disk, floppy disks, and printers attached to both the local and remote S-100 system. We might create the following OSMASTER.GEN file:

```
; OSMASTER.GEN for complex example
STDMASTR ; standard master package
FASLOD   ; non-banked program load
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
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 net circuits
CKTAST = (0000),CKTDRA ; ckt 0 for slaves
         (0100),CKTDRB ; ckt 1 via RS422
DSKAST = 00,DSKDRA ; drv A is local HD
         00,DSKDRB ; drv B is local FD0
         01,DSKDRB ; drv C is local FD1
         80,(0101) ; drv D is remote HD
         81,(0101) ; drv E is remote FD0
         82,(0101) ; drv F is remote FD1
PTRAST = 00,LSTDRA ; ptr A is lcl daisy
         01,LSTDRB ; ptr B is lcl matrix
         80,(0101) ; ptr C is rmt daisy
         81,(0101) ; ptr D is rmt matrix
QUEAST = 00,(0000) ; queue A is local
         00,(0000) ; queue B is local
         80,(0101) ; queue C is remote A
         81,(0101) ; queue D is remote B
DEFDID = (0101)     ; default other master
DSPPAT = 1,2,3,4    ; assgn ptrs to queues
MEMRES = (0400)     ; 1K safety margin
NMBMBS = 0A         ; 10 message buffers
NMBRPS = 5          ; 5 reply packets
NMBSVC = 5          ; 5 server processes
NMBUFS = 14         ; 20 1K disk buffers
```

The generation of the second master operating system could be identical, except that all occurrences of network addresses (0100) and (0101) 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".

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 TurboDOS. If this is your first attempt to generate a TurboDOS system, you may bring up CP/M 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 GEN OSMASTER.SYS, generate an executable system in accordance with the .GEN and .PAR files just constructed. If your hardware has less than 64K installed, don't forget to use the ;Uxxxx option on the GEN command. If your hardware has banked memory, don't forget to use the ;Kxxxx option.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Sysgen Procedure (Continued)

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

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 GEN 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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

(Intentionally left blank.)

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Serialization

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

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

- an origin number that identifies the issuing distributor, and
- a sequential unit number that uniquely identifies each copy of TurboDOS issued by that distributor.

During system generation, the GEN 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.COM) that must be used to add a unique sequential unit number to each copy of TurboDOS issued by the distributor. The GEN 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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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. GEN 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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Example

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

PACKAGE Command

The PACKAGE command lets you combine any collection of relocatable modules into a single concatenated .REL 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 .REL file to be created. If "destfile" is omitted, then the "srcefile" argument is also used as the name of the output .REL file.

If the .PKG file is found, it must contain the list of relocatable modules (.REL files) to be linked together. If the configuration 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:

```
relfile {,relfile}... {;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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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^{R}
   - 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.COM!

4. Using the new serialized disk, use the GEN 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).

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.REL	STDLOADR.REL	STDLOADR.REL
STDSINGL.REL	STDSINGL.REL	STDSINGL.REL
-	STDSPOOL.REL	STDSPOOL.REL
-	-	STDMASTR.REL
-	-	STDSLAVE.REL
-	-	STDSLAVX.REL
FASLOD .REL	FASLOD .REL	FASLOD .REL
BNKMGR .REL	BNKMGR .REL	BNKMGR .REL
CPMSUP .REL	CPMSUP .REL	CPMSUP .REL
MPMSUP .REL	MPMSUP .REL	MPMSUP .REL
RTCNUL .REL	RTCNUL .REL	RTCNUL .REL
PATCH .REL	PATCH .REL	PATCH .REL
SUBMIT .REL	SUBMIT .REL	SUBMIT .REL
OSBOOT .REL	OSBOOT .REL	OSBOOT .REL
-	-	NETLOD .REL
-	-	NETREQ .REL
-	-	NETFWD .REL
-	-	BNKREQ .REL
-	-	MSGFMT .REL
-	-	NETSVC .REL
-	-	QUEMGR .REL
-	-	CONREM .REL
AUTOLOAD.COM	AUTOLOAD.COM	AUTOLOAD.COM
BACKUP .COM	BACKUP .COM	BACKUP .COM
BANK .COM	BANK .COM	BANK .COM
-	-	BATCH .COM
BOOT .COM	BOOT .COM	BOOT .COM
BUFFERS .COM	BUFFERS .COM	BUFFERS .COM
-	-	CHANGE .COM
COPY .COM	COPY .COM	COPY .COM
DATE .COM	DATE .COM	DATE .COM

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Distribution Procedure (Continued)

single-user w/o spooler	single-user with spooler	multi-user networking
DELETE .COM	DELETE .COM	DELETE .COM
DIR .COM	DIR .COM	DIR .COM
DO .COM	DO .COM	DO .COM
DRIVE .COM	DRIVE .COM	DRIVE .COM
DUMP .COM	DUMP .COM	DUMP .COM
ERASEDIR.COM	ERASEDIR.COM	ERASEDIR.COM
-	-	FIFO .COM
FIXDIR .COM	FIXDIR .COM	FIXDIR .COM
FIXMAP .COM	FIXMAP .COM	FIXMAP .COM
FORMAT .COM	FORMAT .COM	FORMAT .COM
GEN .COM	GEN .COM	GEN .COM
LABEL .COM	LABEL .COM	LABEL .COM
-	-	LOGOFF .COM
-	-	LOGON .COM
-	-	MASTER .COM
PRINT .COM	PRINT .COM	PRINT .COM
-	PRINTER .COM	PRINTER .COM
-	QUEUE .COM	QUEUE .COM
-	-	RECEIVE .COM
RELCVT .COM	RELCVT .COM	RELCVT .COM
RENAME .COM	RENAME .COM	RENAME .COM
-	-	SEND .COM
SET .COM	SET .COM	SET .COM
SHOW .COM	SHOW .COM	SHOW .COM
TYPE .COM	TYPE .COM	TYPE .COM
VERIFY .COM	VERIFY .COM	VERIFY .COM

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.

Assembler Notes

Drivers and resident processes for Z80 TurboDOS must be written using a Z80 assembler capable of producing relocatable modules with symbolic linkage information in the industry-standard Microsoft relocatable module format. Both Microsoft's M80 and Digital Research's RMAC assemblers produce object code in this format, and are fine choices for use with TurboDOS.

Another excellent relocatable Z80 assembler is PASM from Phoenix Software Associates. However, PASM produces object modules in a non-standard format.

To make it possible for PASM to be used with TurboDOS, a conversion utility (RELVCT.COM) for converting PASM object modules to standard Microsoft format is furnished with TurboDOS. The command:

```
RELCVT filename
```

converts the specified PASM-format .REL file into Microsoft .REL format. During conversion, the character . is converted to ?, and the character % is converted to @ wherever these characters appear in symbol names.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Assembler Notes (Continued)

Programming examples and driver listings in this document are coded for PASM. If you are used to another assembler, please take note of certain syntax features of PASM which may be different in other assemblers.

Names followed by # are external references to public names defined in other modules. Labels followed :: are public names available for reference in other modules. Some assemblers require such names to be declared using an EXTERN or PUBLIC directive.

Program, data, and common segments are introduced with a .LOC directive. Other assemblers use different directives such as CSEG, DSEG, COMMON, etc. to accomplish the same thing.

Finally, the symbol . represents the current location counter value. Some assemblers use $ or * instead.

Undefined External References

To allow various TurboDOS modules to be included or omitted at will, the GEN command automatically resolves all undefined external references to the default symbol public ?UND? (.UND. using PASM). The common subroutine module COMSUB contains the following subroutine:

```
.UND.:: NOP             ;two bytes of zero
        NOP             ; "    "    "   "
        XRA   A         ;clear A to zero
        RET             ;done
```

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Memory Allocation

The TurboDOS resident occupies the topmost portion of memory in a Z80 system. 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. Memory segments are allocated downward from the base of the TurboDOS resident, reducing the space available for TPA. 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
        LXI   H,36      ;HL=segment size
        CALL  ALLOC#    ;allocate segment
        ORA   A         ;alloc successful?
        JNZ   ERROR     ;NZ -> not enuf mem
        PUSH  H         ;HL=segment address
          :
;code to deallocate a memory segment
        POP   H         ;HL=segment address
        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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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
        .BYTE [128]0   ;contents of node

        .LOC  .PROG.#  ;program segment
;code to add node to end of list
        LXI   H,LSTHED ;HL=head address
        LXI   D,LSTNOD ;DE=node address
        CALL  LNKEND#  ;link to list end

;code to unlink node from list
        LXI   H,LSTNOD ;HL=node address
        CALL  UNLINK#  ;unlink node

;code to add node to beginning of list
        LXI   H,LSTHED ;HL=head address
        LXI   D,LSTNOD ;DE=node address
        CALL  LNKBEG#  ;link to list beg.
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Task Dispatching

TurboDOS incorporates a flexible, efficient mechanism for dispatching the Z80 processor 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
        LXI   H,EVENT   ;HL=semaphore addr
        CALL  WAIT#     ;wait for event

;signal that event has occurred
        LXI   H,EVENT   ;HL=semaphore addr
        CALL  SIGNAL#   ;signal event
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 Z80 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
        LXI   H,6       ;HL=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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Interrupt Service

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

Most interrupt service routines should exit via the usual EI/RETI sequence. However, some periodic interrupt (usually a 50 or 60 hertz clock interrupt) should have an interrupt service routine that exits by jumping to the dispatcher entrypoint ISRXIT# (without enabling interrupts) to provide periodic time-slicing of processes. To avoid excessive dispatcher overhead, don't use ISRXIT# more than about 60 times per second.

It is good programming practice for interrupt service routines to set up an auxilliary stack, in order to avoid the possibility of overflowing the stack area of some transient program. TurboDOS provides a standard interrupt stack area INTSTK# and stack pointer save location INTSP#. A simple interrupt service routine might be coded like this:

```
DEVISR: SSPD   INTSP#    ;save user SP
        LXI    SP,INTSTK# ;SP=aux stack
        PUSH   PSW       ;save registers
        PUSH   B         ;  "      "
        PUSH   D         ;  "      "
        PUSH   H         ;  "      "
        IN     PORT      ;reset interrupt
        LXI    H,EVENT   ;HL=semaphore addr
        CALL   SIGNAL#   ;signal event
        POP    H         ;restore registers
        POP    D         ;  "        "
        POP    B         ;  "        "
        POP    PSW       ;  "        "
        LSPD   INTSP#    ;restore user SP
        EI               ;enable interrupts
        RETI             ;return from int.
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Poll Routines

Devices incapable of interrupting the Z80 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
        LXI   D,POLNOD ;DE=poll node addr
        CALL  LNKPOL#  ;activate poll rtn
        CALL  POLRTN   ;optional pretest
        LXI   H,EVENT  ;HL=semaphore addr
        CALL  WAIT#    ;wait for event
          :

;poll routine signals event when detected
POLNOD: .WORD 0        ;poll rtn linkage
        .WORD 0        ;  "   "     "
POLRTN: IN    PORT     ;get device status
        ANI   MASK     ;did event occur?
        RZ             ;if not, exit
        LXI   H,EVENT  ;HL=semaphore addr
        CALL  SIGNAL#  ;signal event
        LXI   H,POLNOD ;HL=linkage addr
        CALL  UNLINK#  ;unlink poll rtn
        RET            ;all done
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Mutual Exclusion

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

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

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

DRIVER: LXI   H,MXSPH   ;HL=semaphore addr
        CALL  WAIT#     ;wait if in-use
          :

          :
        LXI   H,MXSPH   ;HL=semaphore addr
        CALL  SIGNAL#   ;unlock mut-excl
        RET             ;done
```

Interrupt Status

To permit reliable testing of the interrupt status (enabled or disabled) of the Z80 CPU, TurboDOS provides the subroutine TSTIFF#. It is called with no arguments, and returns with the carry-flag set if and only if interrupts are disabled.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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::LXI   H,MXSPH     ;HL=MX semaph addr
         CALL  WAIT#      ;lock MX
         EI               ;need ints enabled
         LXI   H,RDASPH   ;HL=semaphore addr
         CALL  WAIT#      ;wait data avail
         LDA   CHRSAV     ;get input char
         PUSH  PSW        ;save on stack
         LXI   H,MXSPH    ;HL=MX semaph addr
         CALL  SIGNAL#    ;unlock MX
         POP   PSW        ;return char in A
         RET              ;done
;interrupt service routine
INPISR::SSPD  INTSP#      ;save user's SP
         LXI   SP,INTSTK#  ;SP=aux stack
         PUSH  PSW        ;save registers
         PUSH  B          ;  "      "
         PUSH  D          ;  "      "
         PUSH  H          ;  "      "
         IN    PORT       ;get input char
         STA   CHRSAV     ;save for driver
         LXI   H,RDASPH   ;HL=semaphore addr
         CALL  SIGNAL#    ;signal data avail
         POP   H          ;restore registers
         POP   D          ;   "        "
         POP   B          ;   "        "
         POP   PSW        ;   "        "
         LSPD  INTSP#     ;restore user SP
         EI               ;enable interrupts
         RETI             ;return from int.
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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::LXI   H,MXSPH     ;HL=MX semaph addr
         CALL  WAIT#      ;lock MX
         LXI   D,POLNOD   ;DE=poll rtn node
         CALL  LNKPOL#    ;activate poll rtn
         CALL  POLRTN     ;optional pretest
         LXI   H,RDASPH   ;HL=semaphore addr
         CALL  WAIT#      ;wait data avail
         LDA   CHRSAV     ;get input char
         PUSH  PSW        ;save on stack
         LXI   H,MXSPH    ;HL=MX semaph addr
         CALL  SIGNAL#    ;unlock MX
         POP   PSW        ;return char in A
         RET              ;done
;device poll routine with linkage
POLNOD:  .WORD 0          ;poll rtn linkage
         .WORD 0
POLRTN:  IN    STATUS     ;get device status
         ANI   MASK       ;data available?
         RZ               ;if not, exit
         IN    DATA       ;get input char
         STA   CHRSAV     ;save for driver
         LXI   H,RDASPH   ;HL=semaphore addr
         CALL  SIGNAL#    ;signal data avail
         LXI   H,POLNOD   ;HL=linkage addr
         CALL  UNLINK#    ;unlink poll rtn
         RET              ;done
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Special Segments

In addition to the usual code and data segments, GEN command supports three special location counters (common blocks):

M80/RMAC	PASM	Description
?INIT?	.INIT.#	Initialization code
?PAGE?	.PAGE.#	Page-boundary aligned
?BANK?	.BANK.#	Banked-memory common

?INIT? Segment

In coding driver modules, you will often find a considerable amount of initialization code that is executed only once at cold-start and never needed again. By assembling such code under ?INIT? (.INIT.# using PASM), it will be loaded and executed in lower memory (TPA), and will not occupy space in the resident operating system.

?PAGE? Segment

Sometimes you may need to force a segment of code or data to begin on a 256-byte page boundary. Examples are the simulated CP/M BIOS branch table, and interrupt vectors for Z80 interrupt mode 2. By assembling under ?PAGE? (.PAGE.# using PASM), the segment is guaranteed to be page-aligned.

?BANK? Segment

In banked-memory implementations, you need to be able to place certain code and data in the topmost part of memory which is common to both banks (not switched). Anything assembled under ?BANK? (.BANK.# using PASM) will be assigned to this common region (as specified by the ;Kxxxx option on the GEN command).

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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
        LXI   H,12+4     ;HL=message size+4
        CALL  ALLOC#     ;allocate segment
        PUSH  H          ;save segment addr
          :              ;build msg in seg
        POP   D          ;DE=message addr
        LXI   H,MSGNOD   ;HL=msg node addr
        CALL  SNDMSG#    ;send message

;other process reads/deallocates message
        LXI   H,MSGNOD   ;HL=msg node addr
        CALL  RCVMSG#    ;receive message
        PUSH  H          ;save message addr
          :              ;process message
        POP   H          ;HL=segment addr
        CALL  DEALOC#    ;deallocate seg
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 A
        ORA   A         ;input char avail?
        RZ              ;if not, exit
        CALL  CONIN#    ;get input in A
        CALL  UPRCAS#   ;make upper-case
        MOV   C,A       ;C=character
        CALL  CONOUT#   ;output chr from C

;message output routines
;last char of message has sign-bit set
        CALL  DMS#      ;output following
        .ASCIS "This is a message"
        LXI   H,MSGADR  ;HL=message addr
        CALL  DMSHL#    ;output msg @ HL

;binary-to-decimal output routine
        LXI   H,31416   ;HL=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::.ASCII [0DH] [0AH]
        .ASCII "Implementation by "
        .ASCII "Trigon Computer Corp."
        .ASCII "$"
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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:

```
          .LOC   .INIT.#   ;init code
HDWNIT::  LXI    H,64      ;HL=workspace size
          CALL   ALLOC#    ;alloc workspace
                            ;HL=workspace addr
          LXI    D,MYPROC  ;DE=entrypoint add
          CALL   CRPROC#   ;create process
            :

          .LOC   .PROG.#   ;code segment
MYPROC:   INR    COUNT(Y)  ;increment counter
          LXI    D,60*60   ;1 minute in ticks
          MVI    C,2       ;T-function 2
          CALL   OTNTRY#   ;delay 1 minute
          JMP    MYPROC    ;loop forever
```

CRPROC# automatically allocates a TurboDOS process area (address appears in register IX) 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 HL (as shown above), and will appear to the new process in register IY.

The resident process must make all operating system requests by calling OCNTRY# or OTNTRY# with a C-function or T-function number in register C. It must not call location 0005H or 0050H in the base page, nor make direct calls on kernel routines such as WAIT#, SIGNAL#, DELAY#, SNDMSG#, RCVMSG#, ALLOC#, and DEALOC#.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 X, which Turbodos relies upon as a pointer to its process area. The process may use all other registers as desired.

User-Defined Function

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

Whenever a program invokes T-function 41, TurboDOS transfers control to your USRFCN routine. On entry, register BC contains the address of the 128-byte record area passed from the caller's current DMA address, and registers DE and HL 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 BC at entry) and in any of the registers A-B-C-D-E-H-L.

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#.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 GEN 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@ (DSKDR% using PASM). The GEN command 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 the stack and index registers X and Y, but you may use other registers as desired.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Initialization

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

Your HDWNIT:: routine must not enable interrupts or make calls to WAIT# or DELAY#. In most cases, HDWNIT:: will contain a series of calls to individual driver initialization subroutines contained in other modules.

One-time initialization code that is not needed again should be assembled under the special location counter ?INIT?, so that it doesn't take up space in the resident operating system.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Console Driver

A console driver should be labelled with the public entry symbol CONDR@ (CONDR%:: using PASM). A console number (from CONAST) is passed in register B. The driver must perform a console I/O operation according to the operation code passed in register E:

E-reg	Function
0	Return status in A, char in C
1	Return input character in A
2	Output character passed in C
8	Enter error-message mode
9	Exit error-message mode
10	Conditional output char in C

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

If E=1, the driver obtains an input character (waiting if necessary) and returns it in A.

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

If E=8, the driver prepares to display a TurboDOS error message; if E=9, it reverts to normal. TurboDOS always precedes each error message with an E=8 call and follows it with an E=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 a CR-LF in response to E=8 and E=9 calls.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Console Driver (Continued)

If E=10, the driver determines whether or not it can accept a console output character without dispatching (via WAIT# or DELAY#). If so, it outputs the character passed in C, and returns A=-1 to indicate that the character was accepted. However, if the driver cannot accept a console output character without dispatching, it returns A=0 to indicate that the character was not accepted; TurboDOS will then make an E=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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Printer Driver

A printer driver should be labelled with the public entry symbol LSTDR@ (LSTDR%:: using PASM). A printer number (from PTRAST) is passed in register B. The driver must perform a printer output operation according to the operation code passed in register E:

E-reg	Function
2	Print character passed in C
7	Perform end-of-print-job action

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

If E=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.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Disk Driver

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

```
| Offset |              Contents                |
|                                                |
| ;physical disk request (PDR) packet            |
|  0(X)    .BYTE OPCODE    ;operation code        |
|  1(X)    .BYTE DRIVE     ;drive (base 0)        |
|  2(X)    .WORD TRACK     ;track (base 0)        |
|  4(X)    .WORD SECTOR    ;sector (base 0)       |
|  6(X)    .WORD SECCNT    ;#sectors to rd/wr     |
|  8(X)    .WORD BYTCNT    ;#bytes to rd/wr       |
| 10(X)    .WORD DMAADR    ;DMA addr to rd/wr     |
| 12(X)    .WORD DSTADR    ;DST address           |
| ;copy of disk specification table (DST)        |
| 14(X)    .BYTE BLKSIZ    ;block size (3-7)      |
| 15(X)    .WORD NMBLKS    ;#blocks on disk       |
| 17(X)    .BYTE NMBDIR    ;#directory blocks     |
| 18(X)    .BYTE SECSIZ    ;sector size (0-7)     |
| 19(X)    .WORD SECTRK    ;sectors per track     |
| 21(X)    .WORD TRKDSK    ;tracks on disk        |
| 23(X)    .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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Disk Driver (Continued)

If OPCODE=0, the driver reads SECCNT physical sectors (or equivalently, BYTCNT bytes) into DMAADR, starting at TRACK and SECTOR on DRIVE. The driver returns A=0 if the operation is successful, or A=-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 DMAADR, starting at TRACK and SECTOR on DRIVE. The driver returns A=0 if the operation is successful, or A=-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).

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Disk Driver (Continued)

The driver returns A=-1 if the operation is successful, or A=0 if the drive is not ready or the disk type is unrecognizable. On successful return, TurboDOS moves a copy of the DST into 14(X) through 24(X), where it is available for subsequent operations.

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

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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Bank-Select Driver

Banked-memory systems must include a bank-select driver labelled with the public entry symbol SELBNK::. The function of this routine is simply to select the memory bank (0 or 1) passed in register A. The routine should be coded under the special location counter ?BANK? to ensure it is situated in unswitched common memory. In addition, the SELBNK:: routine must preserve all registers other than A.

All interrupt-driven drivers in a banked-memory system must be designed to service interrupts properly regardless of which bank is active when an interrupt occurs. Drivers for DMA disk controllers must ensure that DMA operations transfer into or out of bank 0 only. Study the sample drivers in the appendix for suggested techniques.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Network Driver

A network circuit driver should be labelled with the public entry symbol CKTDR@ (CKTDR%:: using PASM). A message buffer address is passed in register DE. The driver must either send or receive a network message, according to the operation code passed in register C:

C-reg	Function
0	Receive message into buffer at DE
1	Send message from buffer at DE

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

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

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).

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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
        .BLKB 7         ;registers ACBEDLH
        .BLKB 1         ;user # and flags
        .BLKB 37        ;optional FCB data
        .BLKB 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

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 (C=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 (C=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 A=-1 and DE=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).

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 first word of the downloaded file specifies the base address to which the downloaded system should be moved, and the second word specifies the total byte-length of the system. 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 A.

The entire failure detection, failure recovery, and slave downloading procedure is very hardware-dependent. Study the driver listing in the appendix for guidance.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 B. The driver must perform an I/O operation according to the operation code passed in register E:

E-reg	Function
0	Return input status in A
1	Return input character in A
2	Output character passed in C
3	Set channel baud rate from C
4	Return channel baud rate in A
5	Set modem controls from C
6	Return modem status in A

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

If E=1, the driver obtains an input character (waiting if necessary) and returns it in A.

If E=2, the driver outputs the character passed in C.

If E=3, the driver sets the channel baud rate according to the baud-rate code passed in C. If E=4, the driver returns the channel baud-rate code in A. See T-functions 37 and 38 in the Z80 Programmer's Guide for baud-rate code definitions.

If E=5, the driver sets the modem controls according to the bit-vector passed in C. If E=6, the driver returns the modem status vector in A. See T-functions 39 and 40 in the Z80 Programmer's Guide for bit-vector definitions.

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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: SSPD  INTSP#    ;save user's SP
        LXI   SP,INTSTK# ;SP=aux stack
        PUSH  PSW       ;save registers
        PUSH  B         ;  "       "
        PUSH  D         ;  "       "
        PUSH  H         ;  "       "
        IN    PORT      ;reset interrupt
        CALL  DLYTIC#   ;signal one tick
        LXI   H,RTCCNT  ;get div-by-60 cnt
        DCR   M         ;decrement counter
        JRNZ  ..X       ;not 60 ticks yet
        MVI   M,60      ;reset counter
        CALL  RTCSEC#   ;signal one second
..X:    POP   H         ;restore registers
        POP   D         ;  "       "
        POP   B         ;  "       "
        POP   PSW       ;  "       "
        LSPD  INTSP#    ;restore user's SP
        JMP   ISRXIT#   ;go to dispatcher
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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 8001H     ;Julian date
                        ;base 31-Dec-47
```

Copyright 1984 by Software 2000, Inc.
All rights reserved.

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.COM, and to load and execute it if found. To generate a bootstrap, use the GEN command to combine the standard bootstrap module OSBOOT with your own hardware-dependent driver. Your driver must define the following public entry symbols: INIT, SELECT, READ, XFER, and RAM.

INIT:: is called once to perform any required hardware initialization. It returns with the load base address (where OSLOAD.COM will be loaded) in HL. This address should normally be 0100H, but may have to be higher for a bootstrap ROM in low-memory.

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

READ:: is called to read one physical sector from the last-selected drive. The track is passed in BC, the sector in DE, and the DMA address in HL. It must return A=0 if successful, or A=-1 if an unrecoverable error occurred.

XFER:: is transferred to at the end of the bootstrap process. In most cases, it needs only to set location 0080H to zero (to simulate a null command tail) and jump to 0100H. However, if INIT returned a loader base other than 0100H, then XFER must move the loader down to 0100H before executing it.

RAM:: defines a 64-byte area that OSBOOT can use for working storage. It should not be located where OSLOAD.COM will be loaded!

Copyright 1984 by Software 2000, Inc.
All rights reserved.

(Intentionally left blank.)

APPENDIX

**Sample Driver
Source Listings**

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Sample Driver Source Listings

The remainder of this document consists of assembler source listings of actual drivers. The listings comprise the drivers for a networking TurboDOS system. The master processor is an S-100 single board computer, which incorporates 128K of banked memory, a floppy disk controller (supporting both 5" and 8" drivers), and a pair of RS232 serial ports on-board. The master also has a hard disk controller board connected to a pair of winchester drives. The slave processors are S-100 single-board computers with 128K of banked memory and a pair of RS232 serial ports.

The listings appear in the following order:

Module	Description
EQUATE	common symbolic equates
MPBMAS	master bootstrap driver
NITMAS	master driver initialization
INTMAS	master interrupt handler
BNKMAS	master bank-select driver
CON192	serial console driver, 19.2KB
LSTCTS	serial printer driver, CTS
LSTETX	serial printer driver, ETX/ACK
LSTXON	serial printer driver, XON/XOFF
SPDMAS	master serial/parallel driver
RTCMAS	master clock driver
DSKFDC	master floppy disk driver
DSTFDC	DSTs for 5" and 8" floppy disks
DSKHDC	Winchester hard disk driver
MCDMAS	master circuit driver
NITSLV	slave driver initialization
BNKSLV	slave bank-select driver
SCDSLV	slave circuit driver
RTCSLV	slave clock driver
SPDSLV	slave serial/parallel driver
SLVRES	general slave-reset subroutine

APPENDIX

Sample Driver Source Listings

Copyright 1984 by Software 2000, Inc.
All rights reserved.

Note: Sample driver source listings are available upon request.

(Intentionally left blank.)