Configuration Guide to TurboDOS

September, 1981

NOTE: CP/M, MP/M and CP/NET are trademarks of Digital Research, Inc.

TABLE OF CONTENTS

SECTION 1 — INTRODUCTION

Generating TurboDOS Configurations 1-1
Implementing Driver Modules 1-1
Licensing Requirements 1-2
Serialization 1-3
OEM Responsibilities 1-3
Dealer Responsibilities 1-4
TurboDOS Support 1-4

SECTION 2 — SYSTEM GENERATION

Module Hierarchy 2-2
Process-Level Modules 2-4
Kernel-Level Modules 2-5
Universal Driver-Level Modules 2-7
Hardware-Dependent Driver-Level Modules 2-8
Standard Configurations 2-8
Estimating Memory Requirements 2-10
Linking and Loading 2-11
GEN Command 2-12
Symbolic Patch Facility 2-14
Step-by-Step Procedure for System Generation 2-18
SERIAL Command 2-19
Step-by-Step Procedure for OEM Re-Distribution 2-20

Configuration Guide to TurboDOS

Table of Contents

SECTION 3 — SYSTEM IMPLEMENTATION

Assembler Requirements 3-1
Programming Conventions:
Dynamic Memory Allocation 3-2
Threaded Lists 3-4
Dispatching 3-5
Interrupt Service Routines 3-7
Poll Routines 3-8
Re-Entrancy and Mutual Exclusion 3-9
Sample Interrupt-Driven Device Driver 3-10
Sample Polled Device Driver 3-11
Driver Interface Specifications:
Initialization 3-12
Console Drivers 3-13
Printer Drivers 3-13
Network Drivers 3-14
Disk Drivers 3-15
Real-Time Clock Driver 3-17
Comm Channel Driver 3-18
Bootstrap ROM 3-19

APPENDIX A — Implementation on IMS Equipment A-1

APPENDIX B — Implementation on TRS-80 Model II B-1

APPENDIX C — Sample Driver Listings C-1

INTRODUCTION

This Configuration Guide to TurboDOS provides the information that OEMs, dealers, and sophisticated end-users need to generate various operating system configurations and to implement driver modules for various peripheral components.

A companion document, entitled User's Guide to TurboDOS, provides the information that users need to write and run programs under the TurboDOS operating system. It includes an overview of operating system features, a discussion of architecture and theory of operation, a description of each command, and a definition of each user-callable function.

Generating TurboDOS Configurations

TurboDOS is a modular operating system consisting of more than 40 separate functional modules. These modules are "building blocks" which can be combined in various ways to produce a family of compatible operating systems. TurboDOS configurations include single-task, spooling, time-sharing and networking, with numerous subtle variations possible in each of these broad categories.

Functional modules of TurboDOS are distributed in relocatable form. Hardware-dependent device drivers are packaged in the same fashion. The GEN command is a specialized linkage editor which may be used to combine the desired combination of modules into an executable version of TurboDOS configured with the desired set of functions and device drivers. The GEN command also includes a symbolic patch facility which may be used to alter a variety of operating system parameters.

Section 2 describes each functional module of TurboDOS in detail, illustrates how these modules can be combined in various configurations, and provides step-by-step system generation procedures.

Implementating Driver Modules

TurboDOS has been designed to run on any Z80-based microcomputer with at least 48K of RAM, a random-access mass storage device, and a full-duplex character-oriented console device. The functional modules of TurboDOS are not dependent upon the specific peripheral devices to be used. Rather, a set of hardware-dependent device driver modules must be included in each TurboDOS configuration in order to adapt the operating system to the specific hardware environment.

Typical hardware-dependent device driver modules include:

- Console driver
- Printer driver
- Disk driver
- Network interface driver
- Real-time clock driver
- Communications driver

Although Software 2000 Inc. can supply TurboDOS pre-configured for certain specific hardware configurations, most OEMs and many dealers and end-users will want to implement their own hardware-dependent drivers. Driver modules may be readily written by any competant assembly-language programmer, using a relocating Z80 assembler such as Digital Research's RMAC, Microsoft's MACRO-80, or Phoenix Software Associates' PASM. Section 3 provides detailed instructions to programmers for implementing such driver modules, and the Appendix includes assembly listings of various sample drivers.

Licensing Requirements

TurboDOS is a proprietary software product of Software 2000 Inc. TurboDOS may be used only after the user has paid the required license fee, signed a copy of the TurboDOS software license agreement, and returned the signed agreement to Software 2000 Inc. Then it may be used only in strict conformance with the terms of the software license. Each TurboDOS software license agreement must be filled-out and signed by the end-user (not by an OEM or dealer on his customer's behalf).

Each software license permits the use of TurboDOS only on one specific computer system identified by make, model and serial number. A separate license fee must be paid and a separate license signed for each computer system on which TurboDOS is used. Network slave computers which are also capable of stand-alone operation under TurboDOS must each be licensed separately, but slave computers which cannot be used stand-alone (e.g., because they have no mass storage) do not.

Software 2000 Inc. intends to initiate vigorous legal action against anyone who uses or reproduces TurboDOS software in a manner which is not in strict conformance with the terms of the TurboDOS software license agreement.

Serialization

Each copy of TurboDOS is magnetically serialized with a unique serial number in order to facilitate tracing of unlicensed copies of TurboDOS.

Each relocatable TurboDOS module which is distributed to a dealer or end-user is magnetically serialized with a unique serial number. The serial number consists of two components: an origin number (which identifies the issuing OEM) and a unit number (which uniquely identifies each copy of TurboDOS issued by that OEM). The GEN command verifies that all functional modules which make up a TurboDOS configuration are serialized consistently, and magnetically serializes the resulting executable version of TurboDOS accordingly.

Each relocatable TurboDOS module which is distributed to an OEM is partially serialized with an origin number only. Each OEM is provided with a SERIAL command which must be used to add a unique unit number to the relocatable modules of each copy of TurboDOS issued by that OEM. The GEN command will not accept partially serialized modules that have not been uniquely serialized by the OEM. Conversely, the SERIAL command will not re-serialize modules which have already been fully serialized.

OEM Responsibilities

Each OEM is provided with a master copy of TurboDOS relocatable modules and command processors on diskette. An OEM is authorized to reproduce and distribute copies of TurboDOS to dealers and end-users for use on specifically authorized hardware configurations manufactured or distributed by the OEM. The OEM is required to serialize each copy of TurboDOS with a unique sequential magnetic serial number, and to register each serial number promptly by returning a registration card to Software 2000 Inc. This registration requirement for OEMs is in addition to (not in lieu of) the requirement for licensing of each end-user.

Each OEM is provided with a master copy of TurboDOS documentation in both camera-ready form and in ASCII files on diskette. The OEM is responsible for reproducing the documentation and providing it with each copy of TurboDOS issued by that OEM.

An OEM must require a dealer to sign the TurboDOS dealer agreement and return it to Software 2000 Inc. before the OEM may issue copies of TurboDOS to that dealer. An OEM must require an end-user to sign the TurboDOS software license and return it to Software 2000 Inc. before the OEM may issue a copy of TurboDOS directly to

that end-user.

Dealer Responsibilities

A TurboDOS dealer is permitted to purchase individual serialized copies of TurboDOS software and documentation from Software 2000 Inc. or from an authorized OEM, and to resell them to end-users. Dealers are not authorized to make copies of TurboDOS software or documentation for any purpose whatever.

A TurboDOS dealer must require each end-user to sign the TurboDOS software license and return it to Software 2000 Inc. before issuing a copy of TurboDOS software or documentation to the end-user.

TurboDOS Support

Software 2000 maintains a telephone "hot-line" to provide technical assistance in the use of TurboDOS to its customers. OEMs and dealers should feel free to take advantage of this service whenever technical questions arise concerning the use or configuration of TurboDOS.

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

SYSTEM GENERATION

TurboDOS is a modular operating system consisting of more than 40 separate functional modules. These modules are "building blocks" which can be combined in various ways to produce a family of compatible operating systems. TurboDOS configurations include single-task, spooling, time-sharing and networking, with numerous subtle variations possible in each of these broad categories. This section describes each functional module of TurboDOS in detail, illustrates how these modules can be combined in various configurations, and provides step-by-step system generation procedures.

Functional modules of TurboDOS are distributed in relocatable form. Hardware-dependent device drivers are packaged in the same fashion. The GEN command processor is a specialized linkage editor which may be used to bind together the desired combination of modules into an executable version of TurboDOS configured with the desired set of functions and device drivers. GEN also includes a symbolic patch facility which may be used to alter a variety of operating system parameters.

To simplify the the system generation process, the most commonly used combinations of TurboDOS functional modules are pre-packaged into several standard configurations. Most requirements for TurboDOS can be satisfied by linking the appropriate standard package together with the requisite hardware-dependent drivers.

Module Hierarchy

The flow diagram on the facing page illustrates the functional inter-relationship of TurboDOS modules. As the diagram shows, the software elements of TurboDOS can be viewed as a three-level hierarchy.

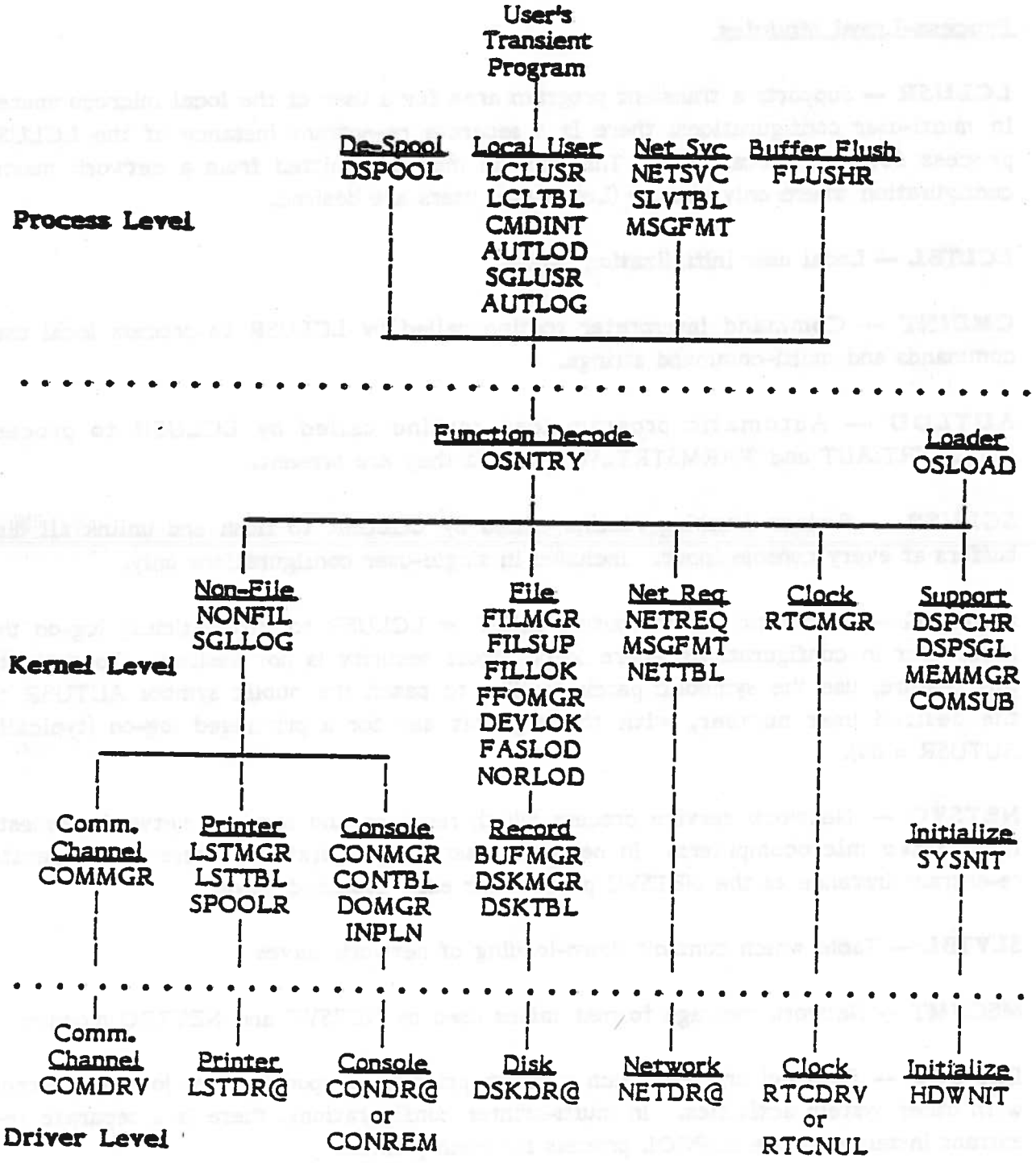
The highest level is known as the "process" level. TurboDOS can support many concurrent processes at this level, and can share the resources of the local computer among them. There are active processes for users who are executing commands and/or transient programs on the local computer. There are also processes for users who are running on remote computers but making network requests of the local computer. There are processes to support de-spooling on each local printer. Finally, there is a process which periodically causes buffered disk records to be flushed (i.e., written out) to disk.

The intermediate level is known as the "kernel" level. The kernel supports the various numbered TurboDOS functions (more than 80 of them), and controls the sharing of microcomputer resources such as processor time, memory, peripheral devices, and disk files. Processes make requests of the kernel through a single entrypoint (OSNTRY) which decodes each function by number and invokes the appropriate module in the kernel.

The lowest level is known as the "driver" level, and contains all of the device-dependent drivers necessary to interface TurboDOS to a particular configuration of microcomputer hardware. Drivers must be provided for each printer, console, disk controller, and network interface. A driver is also required for the real-time clock or other periodic interrupt source (used for time-slicing among processes and for timing of delays). TurboDOS operates most efficiently with interrupt-driven, buffered or DMA-type devices, but can also work satisfactorily with polled and programmed-I/O devices.

The TurboDOS loader OSLOAD.COM is a special program which contains an abbreviated version of the kernel and drivers. Its purpose is to load the full operating system into memory at each system start-up.

All TurboDOS process-level and kernel-level modules permit re-entrant execution in multi-process situations. Most driver-level modules are not re-entrantly coded, and must utilize a mutual-exclusion mechanism to prevent re-entrant execution.



TurboDOS Module Hierarchy

Process-Level Modules

LCLUSR — Supports a transient program area for a user of the local microcomputer. In multi-user configurations, there is a separate re-entrant instance of the LCLUSR process for each local user. This module may be omitted from a network master configuration where only remote (i.e., slave) users are desired.

LCLTBL — Local user initialization tables.

CMDINT — Command interpreter routine called by LCLUSR to process local user commands and multi-command strings.

AUTLOD — Automatic program load routine called by LCLUSR to process COLDSTRT.AUT and WARMSTRT.AUT files if they are present.

SGLUSR — Buffer flushing routine called by LCLUSR to flush and unlink all disk buffers at every console input. Included in single-user configurations only.

AUTLOG — Automatic log-on routine called by LCLUSR to automatically log-on the local user in configurations where logon/logoff security is not desired. To activate this feature, use the symbolic patch facility to patch the public symbol AUTUSR to the desired user number, with the sign-bit set for a privileged log-on (typically AUTUSR = 80).

NETSVC — Network service process which receives and services network requests from slave microcomputers. In network master configurations, there is a separate re-entrant instance of the NETSVC process for each attached slave.

SLVTBL — Table which controls down-loading of network slaves.

MSGFMT — Network message format tables used by NETSVC and NETREQ modules.

DSPOOL — De-spool process which supports printing of spooled print jobs concurrent with other system activities. In multi-printer configurations, there is a separate re-entrant instance of the DSPOOL process for each printer.

FLUSHR — Buffer flusher process which causes memory-resident disk buffers to be flushed (i.e., written out) to disk periodically. Not required in single-user configurations in which SGLUSR is present.

Kernel-Level Modules

OSNTRY — Common kernel entrypoint which decodes each function by number and invokes the appropriate module in the kernel.

FILMGR — File manager which processes requests involving local files. Not required in slave configurations which lack local disk storage.

FILSUP — File support routines required by FILMGR.

FILLOK — Multi-user file interlock routines called by FILMGR. Not required in single-user configurations.

FFOMGR — FIFO management routines called by FILLOK. Not required in single-user configurations.

DEVLOK — Multi-user device interlock routines called by FILMGR. Not required in single-user configurations.

FASLOD — Program load optimizer routine called by FILMGR.

NORLOD — Non-optimized program load routine which may be used instead of FASLOD when memory space is at a premium.

BUFMGR — Buffer manager called by FILMGR. It maintains a pool of memory-resident record buffers used for all record-oriented access to local disk storage.

DSKMGR — Disk manager called by BUFMGR and FASLOD to perform physical accesses to local disk storage.

DSKTBL — Table of disk driver entrypoints and drive-letter-to-disk-number equivalences.

NONFIL — Non-file request manager which handles kernel requests which are not file-oriented.

SGLLOG — Optional module which may be included in multi-user configurations to prevent two or more non-privileged users from logging-on to the same user number concurrently.

CONMGR — Console manager which handles local console input/output.

CONTBL — Table of console driver entrypoints.

DOMGR — DO-file manager which handles activation of DO-files. When a DO-file is active, this module is called by CONMGR to satisfy console input requests from the DO-file.

INPLN — Console input line editor used for buffered console input (function 10), and required by CMDINT.

LSTMGR — List manager which handles local printed output.

LSTTBL — Table of printer driver entrypoints.

SPOOLR — Spooler routine which diverts print output to spool files when the spooler is activated.

COMMGR — Comm channel manager which handles the communications channel.

NETREQ — Network request manager which passes appropriate kernel requests to the network to be satisfied by a network master. Required in network slave configurations.

MSGFMT — Network message format tables used by NETSVC and NETREQ modules. Required in both master and slave network configurations.

NETTBL — Table of network driver entrypoints.

RTCMGR — Real-time clock manager which maintains system date and time.

DSPCHR — Multi-process dispatcher which controls the sharing of local processor time among multiple competing processes.

DSPSGL — Null dispatcher used as an alternative to DSPCHR when only one process is required (e.g., in OSLOAD.COM and in minimal single-user configurations without spooling).

MEMMGR — Memory manager which controls the dynamic allocation and deallocation of memory segments.

COMSUB — Common subroutines required in all configurations.

SYSNIT — System initialization routine which is executed at system start-up.

PATCH — Optional module consisting of 64 bytes of zeroes which may be included to provide space for any required operating system patches.

Universal Driver-Level Modules

RTCNUL — Null real-time clock driver for use in configurations in which there is no periodic interrupt source.

CONREM — Remote console driver for network master to allow access from slave consoles by means of the MASTER command.

Hardware-Dependent Driver-Level Modules

Driver modules are hardware-dependent, and may vary significantly from one TurboDOS implementation to another. In general, the following drivers are required as a minimum:

CONDRV — Console driver allows character-by-character input from a console keyboard and output to a console display. TurboDOS supports multiple console drivers.

LSTDRV — Printer driver allows character-by-character output to a hardcopy peripheral. TurboDOS supports multiple printer drivers.

COMDRV — Comm. channel driver allows character-by-character input and output over one or more communications channels.

DSKDRV — Disk controller driver allows input and output of physical-records on a random-access mass storage device (usually flexible or hard disk). TurboDOS supports multiple disk controller drivers, each of which may support multiple drives.

NETDRV — Network interface driver allows sending and receiving messages to or from a remote microcomputer. TurboDOS supports multiple network interface drivers, each of which may communicate with multiple remote computers.

RTCDRV — Real-time clock driver services interrupts from a periodic interrupt source, used for time-slicing, delay measurement, and updating the system date and time.

HDWNIT — Hardware initialization routine called by SYSNIT. This module usually consists of calls to initialization entrypoints in other drivers.

Standard Configurations

To simplify the the system generation process, the most commonly used combinations of TurboDOS functional modules are pre-packaged into the standard configurations shown in the table on the facing page: STDLOADR, STDSINGL, STDSPOOL, STDMASTR and STDSLAVE. Most requirements for TurboDOS can be satisfied by linking the appropriate standard package together with the requisite driver modules.

Module	Approx. Size (K)	O/S Loader STDLOADR	Single User STDSINGL	Single User w/Spooling STDSPOOL	Network Master STDMASTR	Network Slave STDSLAVE
LCLUSR	.9	-	LCLUSR	LCLUSR	LCLUSR	LCLUSR
LCLTBL	.0	-	LCLTBL	LCLTBL	LCLTBL	LCLTBL
CMDINT	.9	-	CMDINT	CMDINT	CMDINT	CMDINT
AUTLOD	.2	-	AUTLOD	AUTLOD	AUTLOD	AUTLOD
SGLUSR	.1	-	SGLUSR	SGLUSR	-	-
AUTLOG	.0	-	AUTLOG	AUTLOG	-	-
NETSVC	1.1	-	-	-	NETSVC	-
SLVTBL	.0	-	-	-	SLVTBL	-
DSPOOL	.4	-	-	DSPOOL	DSPOOL	-
FLUSHR	.1	-	-	-	FLUSHR	-
OSLOAD	1.2	OSLOAD	-	-	-	-
OSNTRY	.3	-	OSNTRY	OSNTRY	OSNTRY	OSNTRY
FILMGR	1.2	FILMGR	FILMGR	FILMGR	FILMGR	-
FILSUP	2.1	FILSUP	FILSUP	FILSUP	FILSUP	-
FILLOK	.6	-	-	-	FILLOK	-
FFOMGR	.7	-	-	-	FFOMGR	-
DEVLOK	.2	-	-	-	DEVLOK	-
FASLOD	.3	-	FASLOD	FASLOD	FASLOD	-
NORLOD	.1	-	-	-	-	-
BUFMGR	1.0	BUFMGR	BUFMGR	BUFMGR	BUFMGR	-
DSKMGR	.5	DSKMGR	DSKMGR	DSKMGR	DSKMGR	-
DSKTBL	.0	DSKTBL	DSKTBL	DSKTBL	DSKTBL	DSKTBL
NONFIL	.2	-	NONFIL	NONFIL	NONFIL	NONFIL
SGLLOG	.1	-	-	-	-	-
CONMGR	.1	CONMGR	CONMGR	CONMGR	CONMGR	CONMGR
CONTBL	.0	CONTBL	CONTBL	CONTBL	CONTBL	CONTBL
DOMGR	.5	-	DOMGR	DOMGR	DOMGR	DOMGR
INPLN	.1	-	INPLN	INPLN	INPLN	INPLN
LSTMGR	.1	-	LSTMGR	LSTMGR	LSTMGR	LSTMGR
LSTTBL	.0	-	LSTTBL	LSTTBL	LSTTBL	LSTTBL
SPOOLR	.5	-	-	SPOOLR	SPOOLR	-
COMMGR	.1	-	COMMGR	COMMGR	COMMGR	-
NETREQ	1.4	-	-	-	-	NETREQ
MSGFMT	.5	-	-	-	MSGFMT	MSGFMT
RTCMGR	.1	-	RTCMGR	RTCMGR	RTCMGR	-
RTCNUL	.1	RTCNUL	-	-	-	RTCNUL
DSPCHR	.6	-	-	DSPCHR	DSPCHR	-
DSPSGL	.1	DSPSGL	DSPSGL	-	-	DSPSGL
MEMMGR	.3	-	MEMMGR	MEMMGR	MEMMGR	MEMMGR
COMSUB	.3	COMSUB	COMSUB	COMSUB	COMSUB	COMSUB
SYSNIT	.1	-	SYSNIT	SYSNIT	SYSNIT	SYSNIT

Standard TurboDOS Configurations

Estimating Memory Requirements

To estimate memory requirements for a particular TurboDOS configuration, it is necessary to take into account the combined size of functional modules (see table on previous page), hardware-dependent driver modules, disk buffers and other dynamically allocated storage segments.

Hardware-dependent drivers typically require 1K to 3K of memory, depending on the complexity of the hardware involved. Disk buffer space should be as large as possible for optimum performance, especially in a network master. About 4K of disk buffer space is acceptable for a single-user system, although less can be used in a pinch. Other dynamic storage usually doesn't exceed 1K.

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

	O/S Loader STDLOADR	Single User STDSINGL	Single User w/Spooling STDSPOOL	Network Master STDMASTR	Network Slave STDSLAVE
Functional Modules	7K	10K	11K	13K	6K
Device Drivers	2K	2K	2K	3K	1K
Disk Buffer Space	4K	4K	4K	16K	0K
Dynamic Storage	±1K	±1K	±1K	±1K	±1K
Total Memory Req'd	14K	17K	18K	33K	8K
TPA (in 64K system)	n/a	47K	46K	31K	56K

Typical TurboDOS Memory Requirements

Linking and Loading

Functional modules of TurboDOS are distributed in relocatable form. Hardware-dependent device drivers are packaged in the same fashion. The GEN command processor is a specialized linkage editor which may be used to bind together the desired combination of modules into an executable version of TurboDOS configured with the desired set of functions and device drivers. GEN also includes a symbolic patch facility which may be used to alter a variety of operating system parameters.

To generate a TurboDOS system, the GEN command must be used to create both an executable loader OSLOAD.COM and an executable master operating system OSMASTER.SYS. In networking configurations, the GEN command must also be used to create a slave operating system OSSLAVE.SYS. The GEN command can also be used to generate the code for a start-up PROM.

At system start-up, the start-up PROM loads the loader program OSLOAD.COM into the TPA of the master computer and executes it. OSLOAD loads the master operating system OSMASTER.SYS into the topmost portion of memory. In networking configurations, the master operating system down-loads the slave operating system OSSLAVE.SYS into the slave computers on the network.

GEN Command

The GEN command is used for TurboDOS system generation. It links a collection of relocatable modules together into a single executable file. The command format is:

GEN filename1 filename2 ;options

where "filename1" specifies the name of the configuration file (type .GEN) and parameter file (type .PAR) to be used, and "filename2" specifies the name of the executable file (normally type .COM or .SYS) to be created. If "filename2" is omitted from the command line, then "filename1" is used for the executable file and should include an explicit file type (.COM or .SYS).

If the configuration file (type .GEN) is found, it must contain the list of relocatable files to be linked together. If the configuration file is not found, then the GEN command operates in an interactive mode, reading successive directives from the console until terminated by a null directive. The format of each directive (or each line of the configuration file) is:

relfile1, relfile2, ..., relfileN

The GEN command links together all of the specified modules, a two-pass process which displays the name of each module as it is encountered. At the end of the second pass, the GEN command looks for a parameter file (type .PAR) and processes it (if found). Finally, the executable file is written out to disk.

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 OEM) and a unit number (which uniquely identifies each copy of TurboDOS issued by that OEM). The GEN command verifies that all modules to be linked are serialized consistently, and magnetically serializes the resulting executable file accordingly.

The ";options" argument may contain either ";Lxxxx" or ";Uxxxx" to define either the lower or upper boundary of the executable program ("xxxx" is a hexadecimal memory address). The default boundary is ";L0100" if the output file is of type .COM, and ";UFFFF" if the output file is of type .SYS.

The ";options" argument may also contain ";X" to display undefined symbol references (quite normal in TurboDOS system generation), ";M" to print a load map on the printer, and ";S" to print a full symbol table on the printer.

Example:

The following example uses the GEN command to link the modules listed in OSMASTER.GEN and the patch parameters in OSMASTER.PAR, creating the executable file OSMASTER.SYS.

```
0A}GEN OSMASTER.SYS :UBFFF
* STDSINGL, CON192, LSTCTS, SP442
* SER480, BRT4420, RTC442
* DSK401, DSKFMT8, HDWNIT
Pass 1.
LCLUSR    LCLTBL    CMDINT    AUTLOD    SGLUSR    PRVUSR
OSNTRY    FILMGR    FILSUP    FASLOD    BUFMGR    DSKMGR
DSKTBL    NONFIL    CONMGR    CONTBL    DOMGR     INPLN
LSTMGR    LSTTBL    COMMGR    RTCMGR    DSPSGL    MEMMGR
COMSUB    SYSNIT    CON192    LSTCTS    SP442     SER480
BRT442    RTC442    DSK401    DSKFMT    HDWNIT
Pass 2.
LCLUSR    LCLTBL    CMDINT    AUTLOD    SGLUSR    PRVUSR
OSNTRY    FILMGR    FILSUP    FASLOD    BUFMGR    DSKMGR
DSKTBL    NONFIL    CONMGR    CONTBL    DOMGR     INPLN
LSTMGR    LSTTBL    COMMGR    RTCMGR    DSPSGL    MEMMGR
COMSUB    SYSNIT    CON192    LSTCTS    SP442     SER480
BRT442    RTC442    DSK401    DSKFMT    HDWNIT
Processing parameter file:
AUTLOG = 80
NMBUFS = 8
Writing output file.
0A}
```

Symbolic Patch Facility

The GEN command supports a symbolic patch facility which may be used to override various operating system parameters as well as to effect necessary software corrections. Symbolic patches must be stored in a parameter file (type .PAR), which may be built using any ordinary file editor. The format of each .PAR file entry is:

```
location = value1, value2, ..., valueN   ;comments
```

where "value1" through "valueN" are to be loaded into consecutive memory locations starting with "location".

The argument "location" may be a public symbol name, a hexadecimal number, or an expression composed of names and hexadecimal numbers connected by "+" or "-". Hexadecimal numbers must begin with a decimal digit (e.g., "0FFFF"). The location expression must be followed by an equal-sign character.

The arguments "value1" through "valueN" may be expressions (as defined above) or quoted ASCII strings, and must be separated by commas. An expression is stored as a 16-bit word if its value exceeds 255 or if it is enclosed in parentheses; otherwise, an expression 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 the circumflex (e.g., "^X" denotes CTRL-X).

Example:

```
CLBLEN = 9D
NMBUFS = 4
BUFSIZ = 3
CBFCHR = "^F"
CLSCHR = "\"
ATNCHR = "^S"
RESCHR = "^Q"
ABTCHR = "^C"
DSKAST = 00,01,02,03,10,11,12,13,20,21,22,23,30,31,32,33
```

TurboDOS Patch Points

Parameters in TurboDOS which may be useful to patch include the following, shown with their standard values:

In AUTLOD Module:

```
COLDFN  = 0,"COLDSTRTAUT"
                    Cold-start autoload file (12 bytes)
WARMFN  = 0,"WARMSTRTAUT"
                    Warm-start autoload file (12 bytes)
```

In AUTLOG Module:

```
AUTUSR  = 0FF       Automatic log-on user number (sign-bit if privileged)
```

In BUFMGR Module:

```
BUFSIZ  = 3         Default buffer size (0=128, 1=256, 2=512,..., 7=16K)
NMBUFS  = 4         Default number of buffers
```

In CMDINT Module:

```
CLBLEN  = 9D        Command line buffer length
CLSCHR  = "\"       Command line separator character
```

In CONTBL Module:

```
ATNCHR  = "^S"      Attention character
ATNBEL  = "^G"      Attention-received response
RESCHR  = "^Q"      Resume character (attention response)
ABTCHR  = "^C"      Abort character (attention response)
ECOCHR  = "^P"      Echo character (attention response)
PRTCHR  = "^L"      End-print character (attention response)
CONAST  = 00        Console assignment table
CONTBL  = CONDRA    Console driver table
```

In DSKTBL Module:

```
DSKAST  = 00,01,02,03,10,11,12,13,20,21,22,23,30,31,32,33
                    Disk assignment table (16 bytes)
DSKTBL  = DSKDRA,DSKDRB,DSKDRC,DSKDRD
                    Disk driver table (4 words)
```

In FLUSHR Module:

```
BFLDLY  = (012C)    Buffer flush delay in ticks (no flush if zero)
```

In LCLTBL Module:

SPLMOD	= 1	Default spool mode
QUEPTR	= 1	Default spool queue assignment
SPLDRV	= 0FF	Default spool drive 00...0F (system drive if 0FF)

In LCLUSR Module:

MEMRES	= (0100)	Reserved memory between O/S and TPA
SOMSG	= "TurboDOS 1.xx, Copyright (C) 1981, Software 2000, Inc. $"	Sign-on message (56 bytes, must end with "$")

In LSTTBL Module:

LSTAST	= 00,10,20,30	List assignment table (4 bytes)
LSTTBL	= LSTDRA,LSTDRB,LSTDRC,LSTDRD	List driver table (4 words)
NMBQUE	= 1	Number of de-spool queues
DSPPAT	= 1,...,1	De-spool printer assignment table (16 bytes)
NMBPTR	= 1	Number of printers
LSTREM	= 0FF	Default print site (0=local, 0FF=remote)
EOPCHR	= 0	End-of-print character (if nonzero)

In MEMMGR Module:

MEMBLL	= (1103)	Memory base lower limit (standard assures 4K TPA)

In NONFIL Module:

LOGUSR	= 1F	User number for log-off (standard is 31)

In OSLOAD Module:

LOADFN	= 0,"OSMASTERSYS"	Default drive and filename for OSLOAD (12 bytes)
MEMTOP	= (0FFFF)	Top limit of OSLOAD RAM test (don't test if zero)

In SLVTBL Module:

NMBSLV	= 2	Number of network slaves
SLVTBL	= " "	OSSLAVEx.SYS suffix letters (16 bytes)

Explanatory Notes

The patch "AUTUSR = 80" should generally be included in single-user configurations to cause an automatic privileged log-on to user number zero.

The disk assignment table DSKAST contains an array of byte entries corresponding to drives A...P. The high-order nibble of each entry specifies which disk driver (in DSKTBL) to use, while the low order nibble is a drive number passed to the selected driver. In network slaves, the high-order nibble should be set to 15 to indicate a remote drive.

The list assignment table LSTAST contains an array of byte entries corresponding to printers A...P. The high-order nibble of each entry specifies which printer driver (in LSTTBL) to use, while the low order nibble is a printer number passed to the selected driver in the B-register. The console assignment table CONAST works the same way.

If EOPCHR is patched to any non-null ASCII character, then the presence of that character in the print output stream will automatically signal an end-of-print-job condition.

The slave suffix table SLVTBL contains an array of byte entries corresponding to slaves A...P. Each slave operating system is down-loaded from the file "OSSLAVEx.SYS", where "x" is the proper SLVTBL entry. SLVTBL normally contains all spaces, so that all slaves are down-loaded from "OSSLAVE.SYS".

Step-by-Step Procedure for System Generation

To generate a new version of TurboDOS, the following steps may be followed:

1. Bring up a single-user operating system, either CP/M or (preferably) a previous version of TurboDOS. If you are using CP/M, all diskettes will have to be in CP/M-compatible format (one-sided, single-density, 128-byte sector size).

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

3. Using an editor, create or revise the file OSMASTER.GEN containing the names of the relocatable files to be linked together. In most cases, this will consist of the appropriate STDxxxxx file plus all required device drivers.

4. Using an editor, 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 file. If the target machine has less than 64K of memory installed, don't forget to specify a ";Uxxxx" option on the GEN command.

6. If you need to generate a new O/S loader, create or revise the files OSLOAD.GEN and OSLOAD.PAR, and use the command "GEN OSLOAD.COM" to generate an executable loader file.

6. If you need to generate a new slave O/S for a networking configuration, create or revise the files OSSLAVE.GEN and OSSLAVE.PAR, and use the command "GEN OSSLAVE.SYS" to generate an executable down-load file.

7. To test the newly generated system, log onto your working diskette, eject all other diskettes, and enter the command "OSLOAD". If the new system fails to come up or to function properly, you will have to start over at step 1; there is most likely an error in one of your .GEN or .PAR files.

SERIAL Command

Each relocatable TurboDOS module which is distributed to an OEM is partially serialized with an origin number only. Each OEM is provided with a SERIAL command processor which must be used to add a unique unit number to the relocatable modules of each copy of TurboDOS issued by that OEM.

The format of the SERIAL command is:

SERIAL srcfile destfile ;Unnn options

where "srcfile", "destfile" and "options" have exactly the same meanings as in the COPY command, and "nnn" is the unit number expressed as a decimal integer. The SERIAL command works exactly like the COPY command, except that it has the additional function of magnetically serializing .REL files.

The GEN command will not accept partially serialized modules that have not been uniquely serialized by the OEM. Conversely, the SERIAL command will not re-serialize modules which have already been fully serialized.

Example:

```
0A}SERIAL  A:  B:  ;U289 N
A:ASSIGN.COM  copied to B:ASSIGN.COM
.
.
.
A:USER.COM  copied to B:USER.COM
0A}
```

Step-by-Step Procedure for OEM Re-Distribution

To generate a serialized copy of TurboDOS for re-distribution by an OEM to a dealer or end-user, the following steps must be followed:

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

2. Initialize a new diskette, and label it with the TurboDOS version number, the origin and unit numbers, and the required notice "Copyright (C) 1981, Software 2000 Inc.".

3. Using the SERIAL command, copy and serialize the following files from your OEM redistribution master to the new diskette: the appropriate STDxxxxx files, all necessary driver modules, and plus .COM files for AUTOLOAD, BACKUP, BUFFERS, CHANGE, COPY, DATE, DELETE, DIR, DO, DRIVE, DUMP, ERASEDIR, FIFO, FIXMAP, FORMAT, GEN, LABEL, LOGOFF, LOGON, MASTER, PRINT, PRINTER, QUEUE, RECEIVE, RENAME, RESET, SEND, SET, SHOW, TYPE, USER, and VERIFY. Be certain that the new diskette does not contain unserialized modules or SERIAL.COM.

4. Using the new serialized diskette, generate an executable loader and operating system, using the system generation procedure described earlier in this section.

5. In addition to the serialized diskette, the dealer or end-user should receive a TurboDOS start-up PROM and copies of the User's Guide and Configuration Guide.

SYSTEM IMPLEMENTATION

TurboDOS has been designed to run on any Z80-based microcomputer with at least 48K of RAM, a random-access mass storage device, and a full-duplex character-oriented console device. The process-level and kernel-level modules of TurboDOS do not depend upon the specific peripheral devices to be used. Rather, a set of hardware-dependent device driver modules must be included in each TurboDOS configuration in order to adapt the operating system to a particular hardware environment. Device drivers are typically required for consoles, printers, disk controllers, network interfaces, real-time clock, and communications.

Although Software 2000 Inc. can supply TurboDOS pre-configured for certain specific hardware configurations, most OEMs and many dealers and end-users will want to implement their own hardware-dependent drivers. Driver modules may be readily written by any programmer competant in Z80 assembly-language. This section provides detailed instructions to programmers for implementing such driver modules, and the Appendix includes assembly listings of various sample drivers.

Assembler Requirements

Drivers 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 MACRO-80 and Digital Research's RMAC assemblers have these characteristics, and are well suited for implementing TurboDOS drivers.

Phoenix Software Associates' (PSA) assembler (formerly TDL and Xitan) is an excellent relocatable Z80 assembler, but it produces object modules in a non-standard format. To alleviate this problem, a conversion utility (RELCVT.COM) is available from Software 2000 Inc. for converting PSA-format object modules to standard Microsoft format. The command

```
RELCVT  filename
```

converts the PSA-format .REL file specified by "filename" into standard Microsoft .REL format. Wherever the characters "." and "%" appear in names in the PSA-format module, they are replaced by the characters "?" and "@" (respectively) in the Microsoft-format module.

Programming Conventions

Assembly-language examples in this section and in the Appendix are all coded for the PSA assembler. In the examples, the name suffix "/" is used to denote an external name that is defined in another module. The label suffix "::" is used to denote a public name that is available for reference in other modules. Some assemblers require that such names be declared in an EXTERN or PUBLIC statement. Also, the symbol "." represents the current location counter value; some assemblers use "$" or "*" instead.

Dynamic Memory Allocation

The resident portion of TurboDOS resides in the topmost portion of system memory. TurboDOS uses a common memory management module (MEMMGR) to provide dynamic allocation and de-allocation of memory space required for disk buffers, de-spool requests, file interlocks, DO-file nesting, etc. Dynamic memory segments are allocated downward from the base of the TurboDOS resident area, thereby reducing the space available for the transient program area (TPA). Deallocated segments are concatenated with any neighbors and threaded on a free list. A best-fit algorithm is used to reduce memory fragmentation.

Allocation and de-allocation of memory segments is accomplished in this manner:

```
LXI     H,36        ;get size of requested segment in HL
CALL    ALLOC/      ;allocate segment
ORA     A           ;was segment allocated successfully?
JNZ     ERROR       ;if not, error
PUSH    H           ;else, segment base address in HL
.
.
.
POP     H           ;get address of memory segment in HL
CALL    DEALOC/     ;de-allocate segment
```

Note that ALLOC/ clears each newly-allocated segment to zeroes. Note also that ALLOC/ prefixes each dynamic memory segment with a word containing the segment length (including the prefix word itself), so that DEALOC/ can tell how much memory is to be de-allocated.

Sample Interrupt-Driven Device Driver

The following is a simple device driver for an interrupt-driven serial input device. It illustrates the coding techniques described previously:

```
MXLOCK:                         ;mutual-exclusion interlock semaphore
        .WORD   1               ;semaphore count (initialized to 1!)
        .WORD   .               ;semaphore list head forward pointer
        .WORD   .-2             ;semaphore list head backward pointer
;
EVENT:                          ;event semaphore
        .WORD   0               ;semaphore count
        .WORD   .               ;semaphore list forward pointer
        .WORD   .-2             ;semaphore list backward pointer
;
CHRSAV: .BYTE   0               ;input character save location
;
DRIVER: LXI     H,MXLOCK        ;get interlock semaphore address
        CALL    WAIT#           ;wait if driver is already in use
        EI                      ;ensure that interrupts are enabled
        LXI     H,EVENT         ;get event semaphore
        CALL    WAIT#           ;wait for event to occur
        LDA     CHRSAV          ;get input character
        PUSH    PSW             ;save on stack
        LXI     H,MXLOCK        ;get interlock semaphore address
        CALL    SIGNAL#         ;signal driver no longer in use
        POP     PSW             ;return input character in A-register
        RET                     ;done
;
DEVISR: SSPD    INTSP#          ;save user's stack pointer
        LXI     SP,INTSTK#      ;set up auxilliary stack
        PUSH    PSW             ;save all registers
        PUSH    B
        PUSH    D
        PUSH    H
        IN      STATUS          ;get peripheral status
        ANI     MASK            ;is input character available?
        JRZ     ..X             ;if not, exit
        IN      DATA            ;else, get input character
        STA     CHRSAV          ;save input character
        LXI     H,EVENT         ;get event semaphore address
        CALL    SIGNAL#         ;signal that event has occured
..X:    POP     H               ;restore all registers
        POP     D
        POP     B
        POP     PSW
        LSPD    INTSP#          ;restore user's stack pointer
        JMP     ISRXIT#         ;exit through dispacher
```

Re-Entrancy and Mutual Exclusion

All TurboDOS process-level and kernel-level modules permit re-entrant execution by multiple processes. However, most driver-level 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 re-entrant execution.

Using the TurboDOS event semaphore mechanism, such a mutual-exclusion interlock can be implemented very simply in the following manner:

```
MXLOCK:
        .WORD   1               ;mutual-exclusion interlock semaphore
        .WORD   1               ;semaphore count (initialized to 1!)
        .WORD   .               ;semaphore list head forward pointer
        .WORD   .-2             ;semaphore list head backward pointer
;
DRIVER: LXI     H,MXLOCK        ;get interlock semaphore address
        CALL    WAIT#           ;wait if driver is already in use
                .
                .
                .
        LXI     H,MXLOCK        ;get interlock semaphore address
        CALL    SIGNAL#         ;signal driver no longer in use
        RET                     ;done
```

Note that the interlock semaphore count-word must be initialized to 1 (instead of 0) for this scheme to work properly.

Poll Routines

Peripheral devices which are not capable of interrupting the processor must be polled by the device driver. To facilitate this, the TurboDOS dispatcher maintains a threaded list of poll routines, and executes the routines on the list at every dispatch. The function of each poll routine is to check the status of its peripheral device, and to signal the occurrence of an event (e.g., character available, operation complete) when it occurs. The routine LNKPOL# can be called at any time to link a new poll routine onto the poll list.

The only tricky thing about a poll routine is that it must be coded in such a fashion that it will not signal the occurrence a particular event more than once. This can be accomplished in various ways, but a most efficient method is for the poll routine to simply unlink itself from the dispatcher's poll list as soon as it has signalled the occurrence of an event. This can be accomplished in the following manner:

```
EVENT:                          ;event semaphore
        .WORD   0               ;semaphore count
        .WORD   .               ;semaphore list forward pointer
        .WORD   .-2             ;semaphore list backward pointer
          .
          .
          .
        LXI     D,POLNOD        ;get poll routine node address
        CALL    LNKPOL#         ;link poll routine onto poll list
        CALL    POLRTN          ;pre-test peripheral status (optional)
        LXI     H,EVENT         ;get event semaphore address
        CALL    WAIT#           ;wait until event occurs
          .
          .
          .
POLNOD: .WORD   0               ;poll routine node linkage
        .WORD   0
POLRTN: IN      STATUS          ;get peripheral status
        ANI     MASK            ;is input character available?
        RZ                      ;if not, exit
        LXI     H,EVENT         ;else, get event semaphore address
        CALL    SIGNAL#         ;signal that event has occurred
        LXI     H,POLNOD        ;get poll routine node address
        CALL    UNLINK#         ;unlink poll routine from poll list
        RET                     ;done
```

Interrupt Service Routines

The TurboDOS dispatching mechanism is especially efficient when used with interrupt-driven peripheral devices. In most situations, the interrupt service routine simply calls SIGNAL# to indicate that the event associated with the interrupt has occurred.

Service routines for low-frequency interrupts (no more than 100 times per second) should exit by means of the standard interrupt service routine exit ISRXIT# in order to provide frequent time-slicing of processes. Service routines for high-frequency interrupts (occurring more than 100 times per second) should simply enable interrupts and return, in order to avoid excessive dispatch overhead.

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 a user's program. TurboDOS provides a standard interrupt stack area (INTSTK#) and stack pointer save location (INTSP#) for this purpose.

A simple interrupt service routine for a low-frequency interrupt could be coded in this manner:

```
DEVISR: SSPD    INTSP#      ;save user's stack pointer
        LXI     SP,INTSTK#  ;set up auxilliary stack
        PUSH    PSW         ;save all registers
        PUSH    B
        PUSH    D
        PUSH    H
        IN      STATUS      ;reset the interrupt condition
        LXI     H,EVENT     ;get event semaphore address
        CALL    SIGNAL#     ;signal that event has occured
        POP     H           ;restore all registers
        POP     D
        POP     B
        POP     PSW
        LSPD    INTSP#      ;restore user's stack pointer
        JMP     ISRXIT#     ;exit through dispacher
```

In more complex interrupt situations, it may be necessary for an interrupt service routine to determine which of several possible events occurred, and to signal one of several alternative semaphores. Sometimes it may be desirable for an interrupt service routine to perform a data buffering function (e.g., to provide keyboard type-ahead).

If the occurrence of an event is signalled but no process is waiting for it, then SIGNAL# simply 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 (e.g., head-settle delay, carriage-return delay) rather than for the occurrence of a specific event. The TurboDOS dispatcher provides a delay facility (DELAY#) which permits other processes to use the microprocessor while one process is waiting for such a time interval to expire. Delay intervals are measured in an implementation-defined unit called a "tick"; in most implementations, ticks occur 50 or 60 times per second. Delays may be coded in the following manner:

```
        .
        .
        .
LXI     H,6         ;get number of ticks to delay
CALL    DELAY#      ;delay for specified interval
        .
        .
        .
```

A delay of zero ticks may be specified to effect a very short delay, or simply to relinquish the processor to other processes on a "courtesy" basis.

For best performance, all driver delays should be accomplished by means of WAIT# (wait for an event to be signalled) or DELAY# (wait for a given interval of time to elapse). Drivers should never be coded to spin in a wait loop.

Dispatching

TurboDOS incorporates an extremely efficient and flexible mechanism for dispatching the Z80 microprocessor among various competing processes. In writing device drivers for TurboDOS, the programmer must take extreme care to use the dispatcher correctly in order to attain maximum performance.

Basically, the dispatcher enables one process to wait for some event (e.g., character available, operation complete) while allowing other processes to utilize the microprocessor. For each such event, the programmer must define a three-word structure called an "event semaphore". A semaphore consists of a count-word followed by a two-word list head. The count-word is used by the dispacher to keep track of the status of the event, while the list head defines a threaded list of processes waiting for the event.

There are two fundamental operations which affect an event semaphore: waiting for the event to occur (WAIT#), and signalling that the event has occurred (SIGNAL#). These are coded in the following manner:

```
EVENT:                          ;event semaphore
        .WORD   0               ;semaphore count
        .WORD   .               ;semaphore list forward pointer
        .WORD   .-2             ;semaphore list backward pointer
          .
          .
          .
        LXI     H,EVENT         ;get event semaphore address
        CALL    WAIT#           ;wait until event occurs
          .
          .
          .
        LXI     H,EVENT         ;get event semaphore address
        CALL    SIGNAL#         ;signal that event has occured
```

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

Threaded Lists

All dynamic structures in TurboDOS are maintained as threaded lists with bidirectional linkages. This technique permits a node to be easily added or deleted anywhere in a threaded list without searching. The list head and each list node must contain a two-word linkage (forward pointer and backward pointer).

Manipulation of threaded lists is accomplished in this manner:

```
LSTHED:                             ;list head (initialized to empty)
        .WORD   .                   ;forward pointer
        .WORD   .-2                 ;backward pointer
;
LSTNOD:                             ;list node
        .WORD   0                   ;forward pointer
        .WORD   0                   ;backward pointer
        .BLKB   128                 ;node body
          .
          .
          .
        LXI     H,LSTHED            ;get list head address in HL
        LXI     D,LSTNOD            ;get new node address in DE
        CALL    LNKEND#             ;link node to end of list
          .
          .
          .
        LXI     H,LSTNOD            ;get node address in HL
        CALL    UNLINK#             ;unlink node from list
          .
          .
          .
        LXI     H,LSTHED            ;get list head address in HL
        LXI     D,LSTNOD            ;get new node address in DE
        CALL    LNKBEG#             ;link node to beginning of list
```

In programming hardware-dependent driver modules, it is frequently necessary to include a considerable amount of initialization code which is executed only once (at system start-up) and never needed again. Using DEALOC#, the memory space occupied by such initialization code can be made available to satisfy subsequent dynamic memory requirements. To do this, the code segment must be prefixed with a word containing the segment length:

```
          .WORD   LENGTH+2    ;length to be de-allocated
;
HDWNIT:: XRA      A           ;start of initialization code
           .
           .
           .
          LXI     H,HDWNIT    ;get beginning of segment
          JMP     DEALOC#     ;de-allocate segment
;
LENGTH    =       $-HDWNIT    ;length of segment
```

Sample Polled Device Driver

The following is a simple device driver for a polled serial input device. It illustrates the coding techniques described previously:

```
MXLOCK:
        .WORD   1       ;mutual-exclusion interlock semaphore
        .WORD   .       ;semaphore count (initialized to 1!)
        .WORD   .-2     ;semaphore list head forward pointer
                        ;semaphore list head backward pointer
;
EVENT:                  ;event semaphore
        .WORD   0       ;semaphore count
        .WORD   .       ;semaphore list forward pointer
        .WORD   .-2     ;semaphore list backward pointer
;
CHRSAV: .BYTE   0       ;input character save location
;
DRIVER: LXI     H,MXLOCK ;get interlock semaphore address
        CALL    WAIT#   ;wait if driver is already in use
        LXI     D,POLNOD ;get poll routine node address
        CALL    LNKPOL# ;link poll routine onto poll list
        CALL    POLRTN  ;pre-test peripheral status (optional)
        LXI     H,EVENT ;get event semaphore address
        CALL    WAIT#   ;wait until event occurs
        LDA     CHRSAV  ;get input character
        PUSH    PSW     ;save on stack
        LXI     H,MXLOCK ;get interlock semaphore address
        CALL    SIGNAL# ;signal driver no longer in use
        POP     PSW     ;return input character in A-register
        RET             ;done
;
POLNOD: .WORD   0       ;poll routine node linkage
        .WORD   0
POLRTN: IN      STATUS  ;get peripheral status
        ANI     MASK    ;is input character available?
        RZ              ;if not, exit
        IN      DATA    ;else, get input character
        STA     CHRSAV  ;save input character
        LXI     H,EVENT ;else, get event semaphore address
        CALL    SIGNAL# ;signal that event has occurred
        LXI     H,POLNOD ;get poll routine node address
        CALL    UNLINK# ;unlink poll routine from poll list
        RET             ;done
```

Driver Interface Specifications

The interface specifications for various kinds of device drivers are described below. Drivers may be packaged into as many or few separate modules as desired by the programmer. In general, it is easier to reconfigure TurboDOS for a wide variety of peripheral devices if the driver for each device is packaged as a separate module.

TurboDOS may be configured with multiple disk, console, printer and network drivers. The disk driver entrypoint table refers to disk driver entrypoints DSKDRA#, DSKDRB#, DSKDRC#, etc. Each disk driver should be coded with a public entrypoint DSKDR@:: (or DSKDR%:: if PSA assembler and RELCVT are used). The GEN command automatically maps successive definitions of such names by replacing the trailing @ by A, B, C, etc. The same technique should be used for console, printer, and network drivers.

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 ?UND?#. The TurboDOS common subroutine module COMSUB contains the following stub routine:

```
?UND?::  NOP          ;single- or double-length load
         NOP          ;of undefined returns zero
         XRA   A      ;call of undefined returns A=0
         RET          ;done
```

Thus, it is always safe to load or call an external name, whether or not it is defined.

Driver routines must preserve the stack and the index registers X and Y, but may use other registers as desired.

Initialization

All necessary hardware initialization and interrupt vector setup should be performed by an initialization routine that begins with the public entry name HDWNIT::. This routine is called by TurboDOS at system start-up with interrupts disabled. The hardware initialization procedure must not enable interrupts or make calls to WAIT# or DELAY#. In most cases, the HDWNIT:: routine should contain a series of calls to individual driver initialization subroutines.

Console Drivers

Each console driver routine should begin with the public entry name CONDR@::, and should perform a console operation in accordance with the operation code (0, 1, 2, 8 or 9) passed by TurboDOS in the E-register. A console number is passed in the B-register (obtained from the least-significant nibble of the console assignment table entry CONAST#).

If E=0, the driver must determine if a console input character is available. It must return with A=-1 if a character is available, or with A=0 if no character is available. If a character is available, the driver must return it in the C-register, but must not "consume" the character. (This look-ahead capability is used by TurboDOS to detect attention requests.)

If E=1, the driver must obtain a console input character (waiting for one if necessary), and return it in the A-register.

If E=2, the driver must output to the console the character passed by TurboDOS in the C-register.

If E=8, the driver should prepare to display a TurboDOS error message; if E=9, the driver should revert to normal display. Error message displays issued by TurboDOS are always preceded by an E=8 call and followed by an E=9 call. This gives the console driver the opportunity to take special action for system error messages (e.g., 25th line, reverse video). For simple console devices, the driver should perform a carriage-return and line-feed in response to E=8 and E=9 calls.

Printer Drivers

Each printer driver routine should begin with the public entry name LSTDR@::, and should perform a printer operation in accordance with the operation code (2 or 7) passed by TurboDOS in the E-register. A printer number is passed in the B-register (obtained from the least-significant nibble of the printer assignment table entry LSTAST#).

If E=2, the driver must output to the printer the character passed by TurboDOS in the C-register.

If E=7, the driver should take any appropriate end-of-print-job action (e.g., re-align forms, drop ribbon, home print head).

Network Drivers

(To be supplied in a future revision.)

Disk Drivers

Each disk driver routine should begin with the public entry name DSKDR@::, and should perform a physical disk operation as specified by the physical disk request packet whose address is passed by TurboDOS in the X-register. The format of the physical disk request packet is:

```
X+0:    .BYTE   OPCODE      ;disk operation code
X+1:    .BYTE   DRIVE       ;drive number on controller (base 0)
X+2:    .WORD   TRACK       ;physical track number (base 0)
X+4:    .WORD   SECTOR      ;physical sector number (base 0)
X+6:    .WORD   SECCNT      ;number of sectors to read or write
X+8:    .WORD   BYTCNT      ;number of bytes to read or write
X+10:   .WORD   DMAADR      ;DMA address for read or write
X+12:   .WORD   DSTADR      ;disk specification table address
;
;copy of disk specification table follows
;
X+14:   .BYTE   BLKSIZ      ;block size (3=1K, 4=2K,..., 7=16K)
X+15:   .WORD   NMBLKS      ;number of blocks, total
X+17:   .BYTE   NMBDIR      ;number of directory blocks
X+18:   .BYTE   SECSIZ      ;sector size (0=128, 1=256, 2=512,..., 7=16K)
X+19:   .WORD   SECTRK      ;sectors per track
X+21:   .WORD   TRKDSK      ;total tracks on disk
X+23:   .WORD   RESTRK      ;reserved tracks on disk
```

If OPCODE=0, then the driver must read SECCNT physical sectors (or BYTCNT bytes) into DMAADR, starting at TRACK and SECTOR on DRIVE. Return with A=-1 if an unrecoverable error occurs, otherwise return with A=0. Although TurboDOS may request many consecutive sectors to be read, it will never request an operation which extends past the end of the specified track.

If OPCODE=1, then the driver must write SECCNT physical sectors (or BYTCNT bytes) from DMAADR, starting at TRACK and SECTOR on DRIVE. Return with A=-1 if an unrecoverable error occurs, otherwise return with A=0. Although TurboDOS may request many consecutive sectors to be written, it will never request an operation which extends past the end of the specified track.

If OPCODE=2, then the driver must determine the type of disk mounted in the specified drive, and must return in DSTADR the address of an 11-byte disk specification table structured as follows:

```
DST:    .BYTE   BLKSIZ      ;block size (3=1K, 4=2K,..., 7=16K)
        .WORD   NMBLKS      ;number of blocks, total
        .BYTE   NMBDIR      ;number of directory blocks
        .BYTE   SECSIZ      ;sector size (0=128, 1=256, 2=512,..., 7=16K)
        .WORD   SECTRK      ;sectors per track
        .WORD   TRKDSK      ;total tracks on disk
        .WORD   RESTRK      ;reserved tracks on disk
```

On return, TurboDOS moves a copy of the disk specification table into X+14 through X+24, where it is available for subsequent read and write operations on that drive. If the drive is not ready or the type is unrecognizable, the driver must return A=0, otherwise it must return A=-1.

If OPCODE=3, then the driver must determine whether or not the specified drive is ready. Return A=-1 if the drive is ready, otherwise return A=0.

If OPCODE=4, then the driver must format (i.e., initialize) the specified TRACK on DRIVE. Hardware-dependent formatting information will be provided at DMAADR. Return with A=-1 if an unrecoverable error occurs, otherwise return with A=0.

Real-Time Clock Driver

The real-time clock driver normally consists of an interrupt service routine which responds to interrupts from a periodic interrupt source (preferably 50 to 60 times per second). The interrupt service routine should call DLYTIC# once per system tick to synchronize process delay requests. It should also call RTCSEC# once per second (i.e., every 50 or 60 ticks) to update the system time and date. Finally, it should exit through ISRXIT# to provide a periodic system time-slice.

Excluding necessary initialization code, a typical real-time clock driver might look like this:

```
RTCCNT: .BYTE   1           ;divide-by-60 counter
;
RTCISR: SSPD    INTSP#      ;save user's stack pointer
        LXI     SP,INTSTK#  ;set up auxilliary stack
        PUSH    PSW         ;save all registers
        PUSH    B
        PUSH    D
        PUSH    H
        IN      STATUS      ;reset the interrupt condition
        CALL    DLYTIC#     ;signal one tick elapsed time
        LXI     H,RTCCNT    ;get divide-by-60 counter
        DCR     M           ;decrement counter
        JRNZ    ..X         ;not 60 ticks yet, exit
        MVI     M,60        ;else, reset counter to 60 ticks
        CALL    RTCSEC#     ;signal one second elapsed time
..X:    POP     H           ;restore all registers
        POP     D
        POP     B
        POP     PSW
        LSPD    INTSP#      ;restore user's stack pointer
        JMP     ISRXIT#     ;exit through dispacher
```

If it is possible to determine the date and/or time-of-day at cold-start (e.g., by means of a battery-powered clock board), then the driver may initialize the following public symbols in RTCMGR:

```
SECS::  .BYTE   0           ;0...59
MINS::  .BYTE   0           ;0...59
HOURS:: .BYTE   0           ;0...23
JDATE:: .WORD   8001H       ;Julian date, based 31 Dec 47
```

Comm Channel Drivers

The comm channel driver supports the TurboDOS communications extensions (functions 87...93), and is not required if these functions are not used. The comm channel driver routine should begin with the public entry name COMDRV::, and should perform a comm channel operation in accordance with the operation code passed by TurboDOS in the E-register. A channel number is passed in the B-register.

If E=0, the driver must determine if an input character is available on the specified channel. It must return with A=-1 if a character is available, or with A=0 if no character is available.

If E=1, the driver must obtain an input character from the specified channel (waiting for one if necessary), and return it in the A-register.

If E=2, the driver must output to the specified channel the character passed by TurboDOS in the C-register.

If E=3, the driver must set the baud rate of the specified channel according to the baud rate code passed by TurboDOS in the C-register. (See function 90 in the User's Guide for definition of the codes.)

If E=4, the driver must obtain the current baud rate code for the specified channel, and return it in the A-register.

If E=5, the driver must set the modem controls of the specified channel according to the modem control vector passed by TurboDOS in the C-register. (See function 92 in the User's Guide for definition of the vector.)

If E=6, the driver must obtain the current modem status vector for the specified channel, and return it in the A-register. (See function 93 in the User's Guide for definition of the vector.)

Bootstrap ROM

Implementation of a TurboDOS bootstrap ROM involves linking the standard bootstrap module OSBOOT with a hardware-dependent driver OSBDRV. This should be accomplished with the GEN command, using the ";Lxxxx" option to establish the desired ROM base address. Since the OSBOOT module requires only 0.4K, the completed bootstrap can fit in a 1K ROM (e.g., 2708) if the driver is kept simple enough. The driver module OSBDRV must define five public entry names: INIT::, SELECT::, READ::, XFER::, and RAM::.

INIT:: is called at the beginning of the bootstrap process, and performs any required hardware initialization (e.g., of the disk controller). It must return with the load base address in the HL-registers. The load base address determines the RAM where loading of the file OSLOAD.COM will begin. It should normally be 0100H, but may have to be a higher address if low RAM cannot be written while the ROM is enabled.

SELECT:: selects the disk drive according to the drive number 0...15 passed in the A-register. If the selected drive is not ready or non-existent, then this routine must return A=0. Otherwise, it must return A=-1, and must return the address of an appropriate disk specification table in the HL-registers. The disk specification table is an 11-byte table whose format is the same as described earlier for the normal disk driver.

READ:: reads one physical sector from the last selected drive into RAM. On entry, the physical track is passed in the BC-registers, the physical sector is passed in the DE-registers, and the starting RAM address is passed in the HL-registers. The routine must return with A=0 if the operation was successful, or with A=-1 if an unrecoverable error occurred.

XFER:: is executed at the end of the bootstrap process, and transfers control to the loader program OSLOAD.COM which has been loaded into RAM. In most cases, this involves simply setting location 0080H to zero (to simulate a null command tail), and jumping to 0100H. However, if INIT returned a loader base other than 0100H, then XFER should move the loader program down to 0100H prior to execution.

RAM:: defines the beginning of a 64-byte area of RAM that OSBOOT can use as working storage. Obviously, it should not be located in the area in which OSLOAD.COM will be loaded!